

FIG. 1

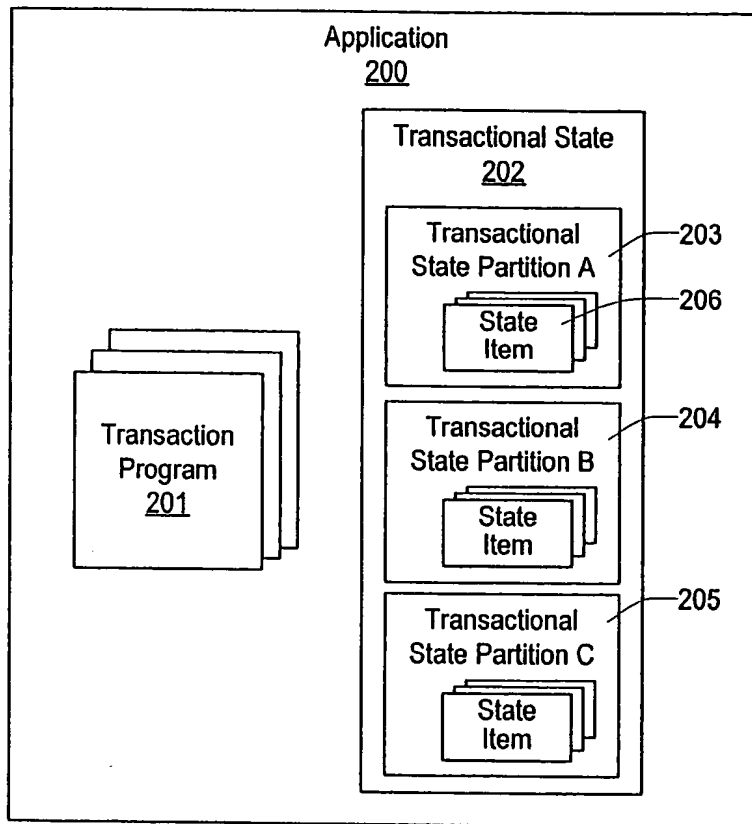


FIG. 2

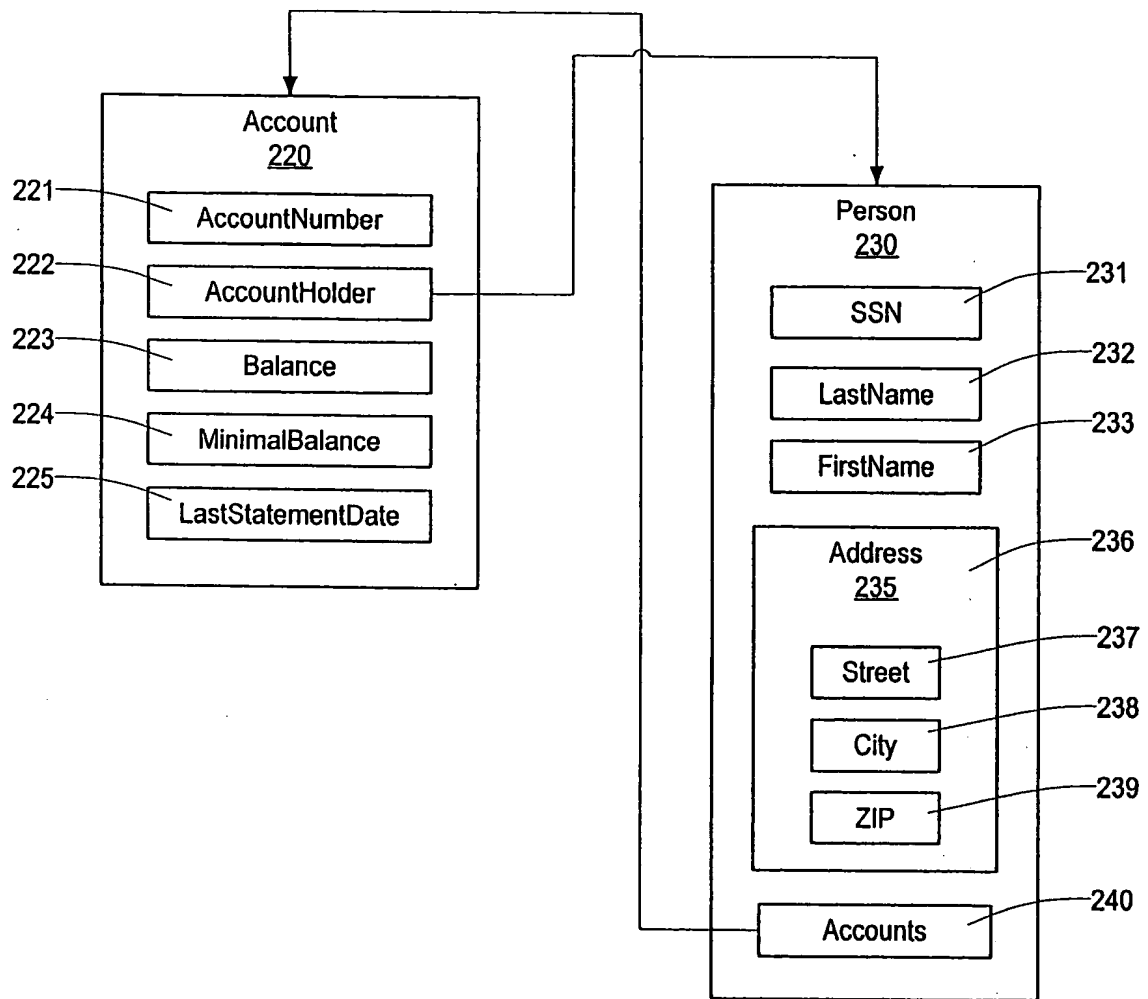


FIG. 2-A

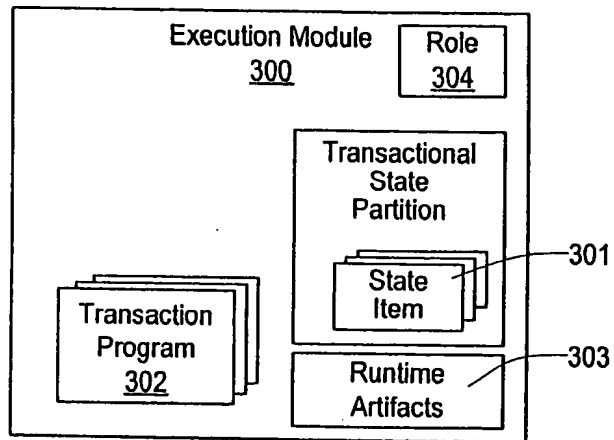


FIG. 3

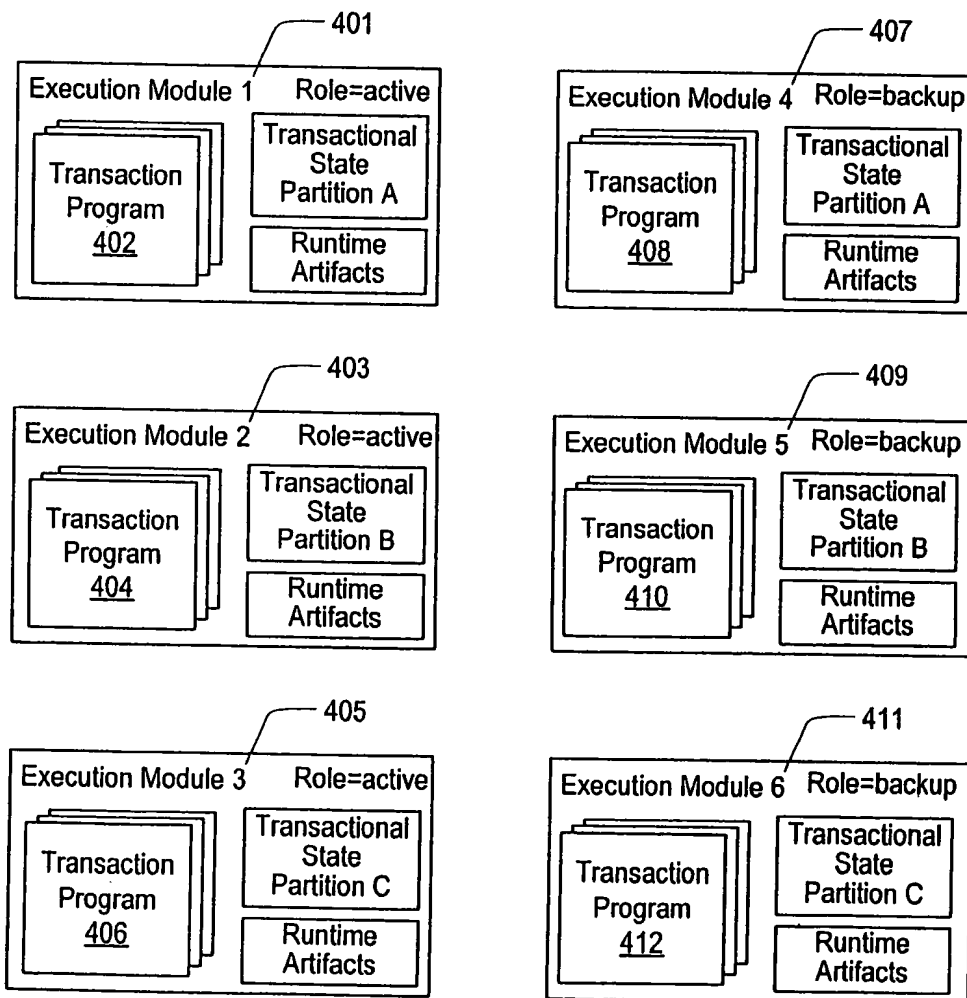


FIG. 4

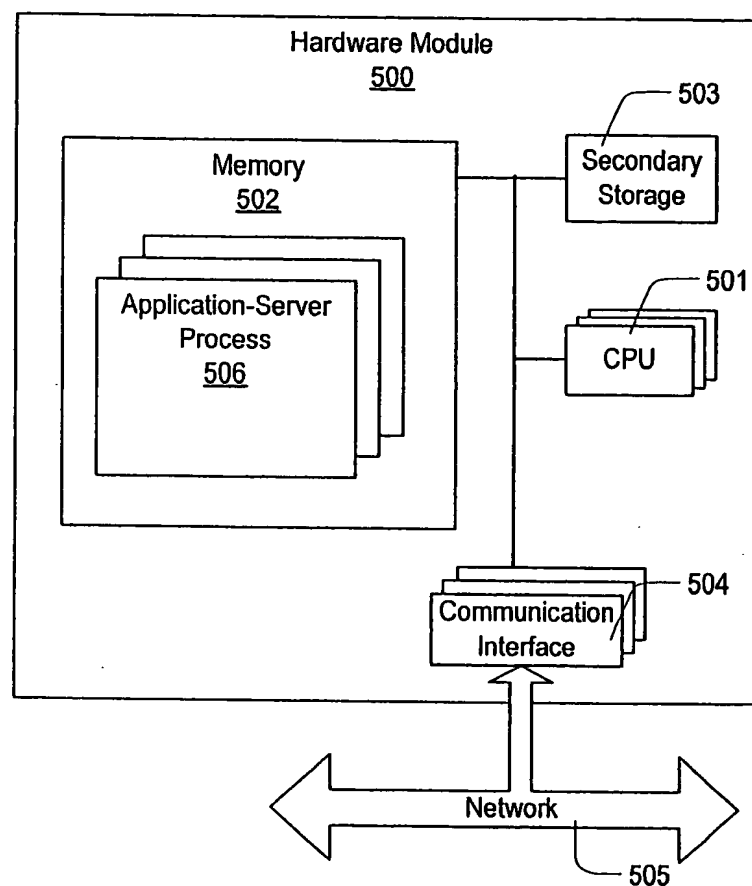


FIG. 5

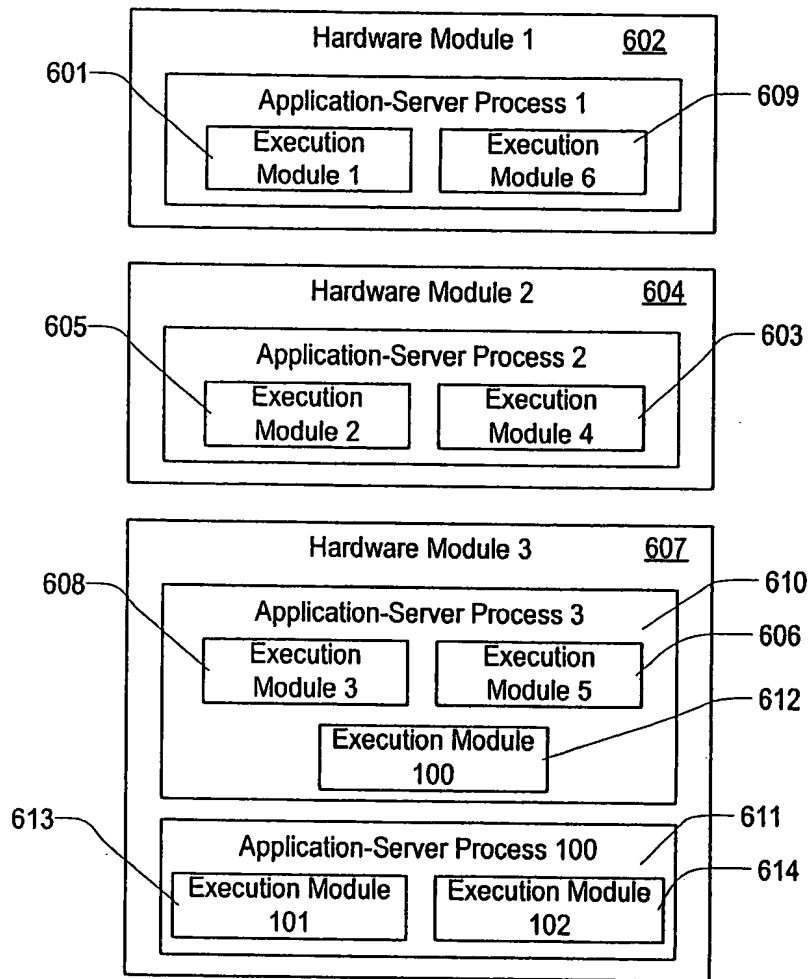


FIG. 6

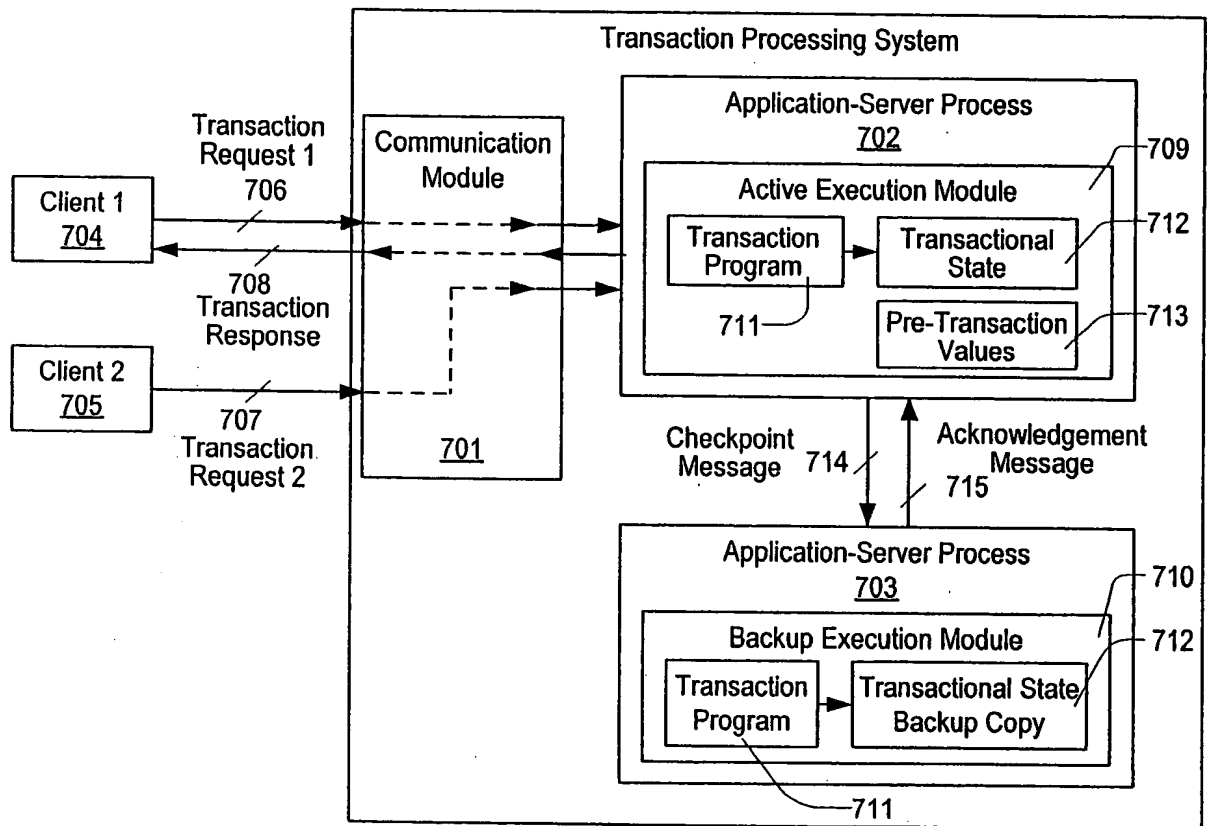


FIG. 7

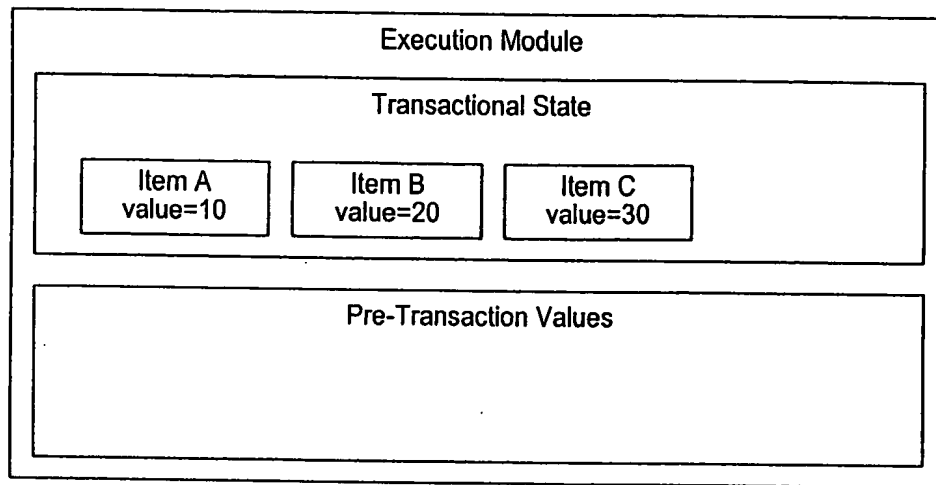


FIG. 8

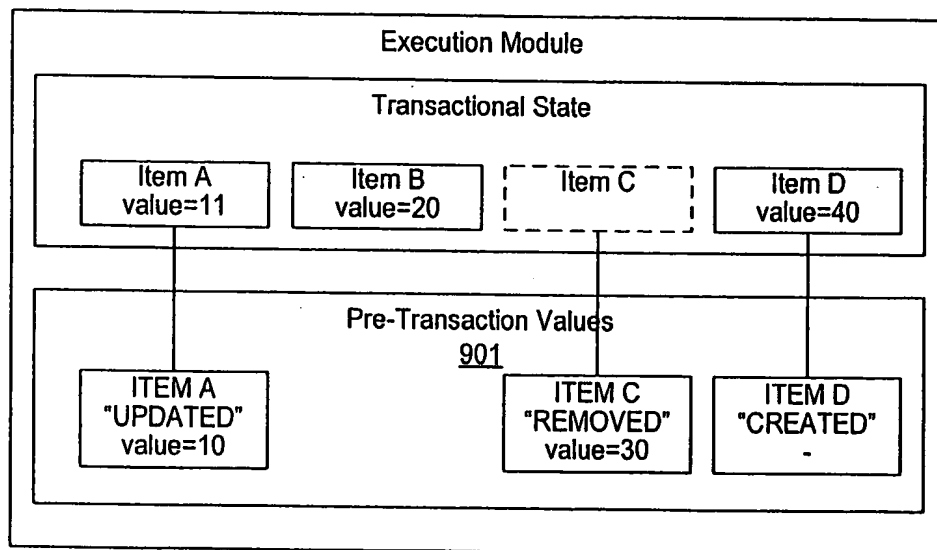


FIG. 9



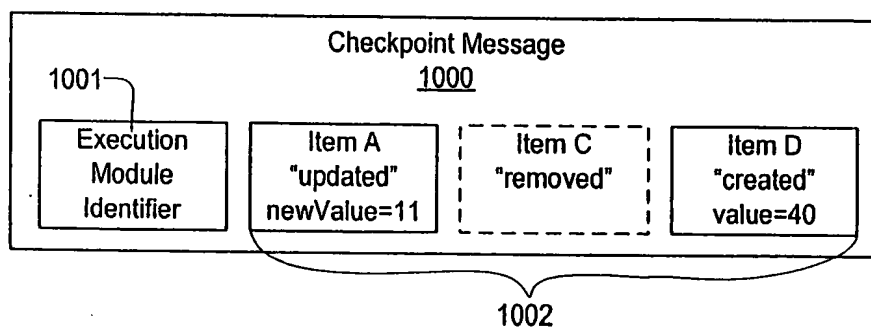


FIG. 10

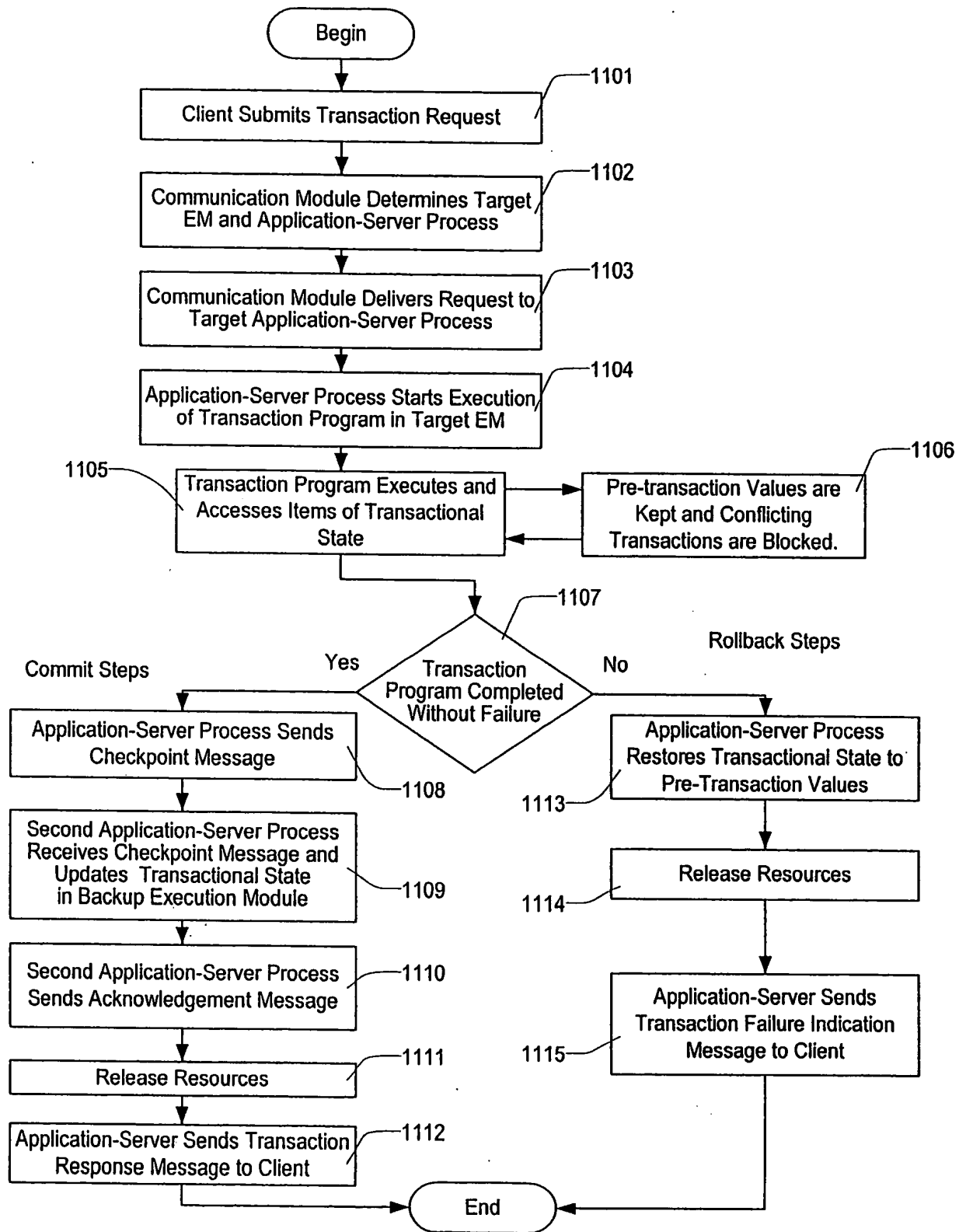


FIG. 11

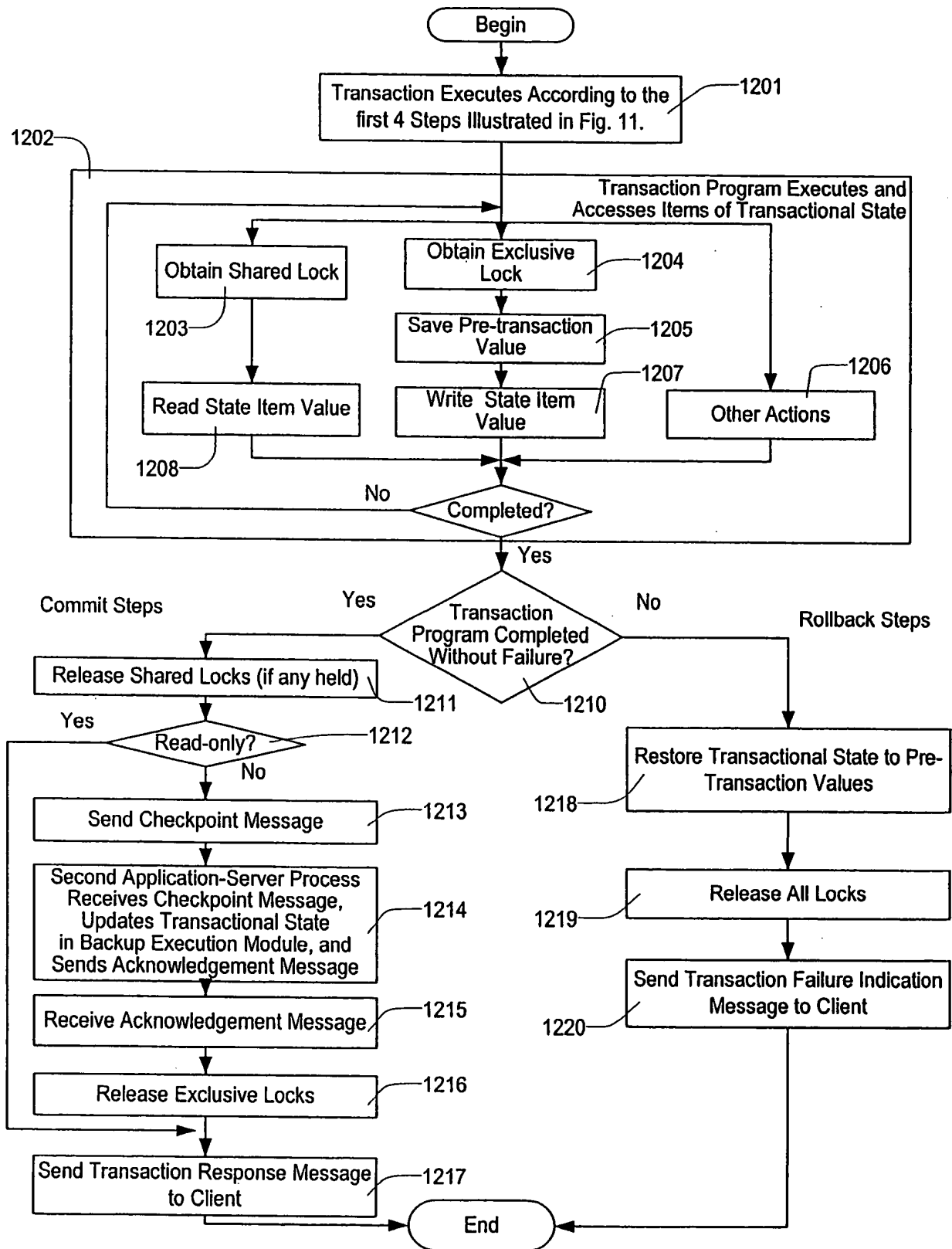


FIG. 12

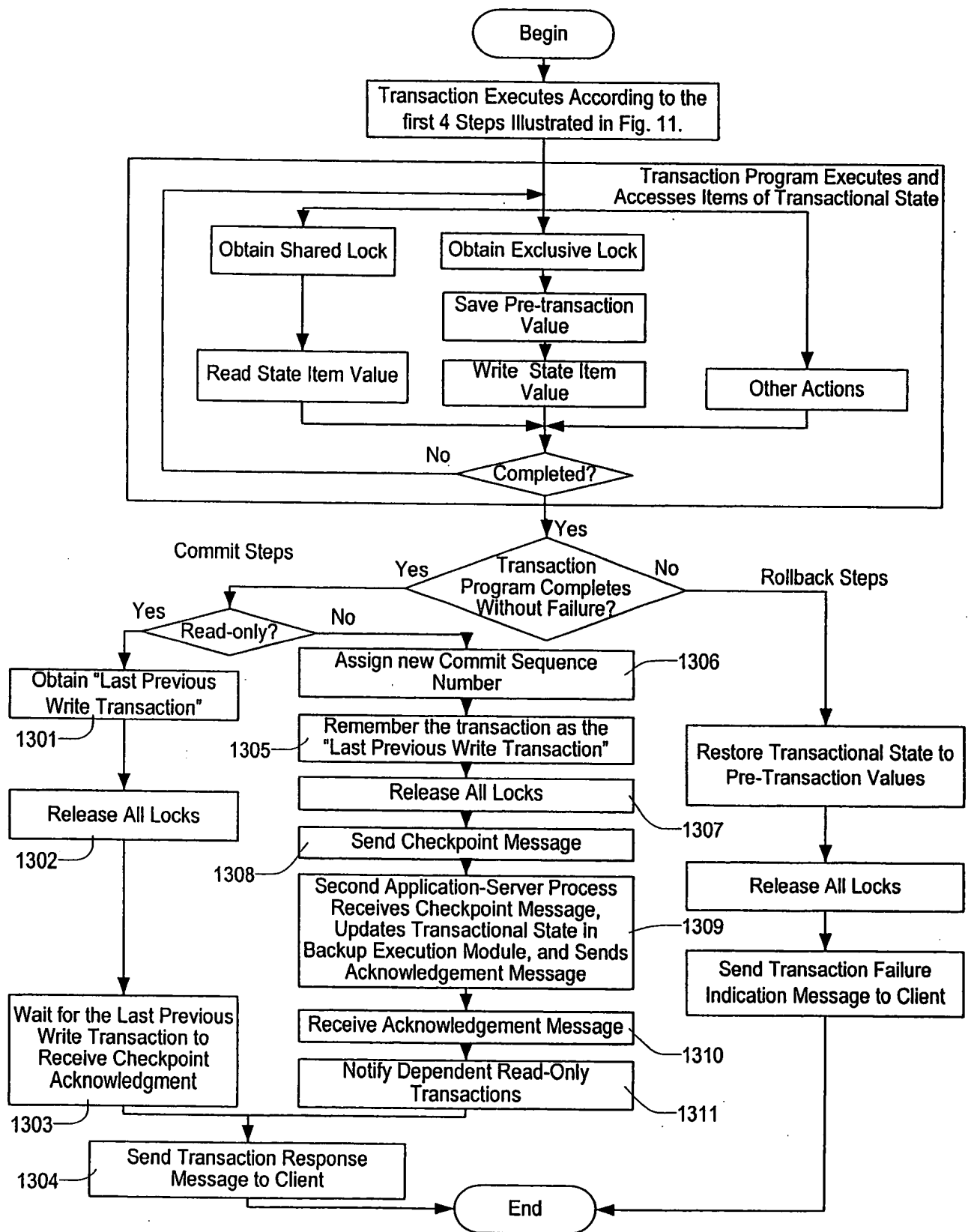


FIG. 13

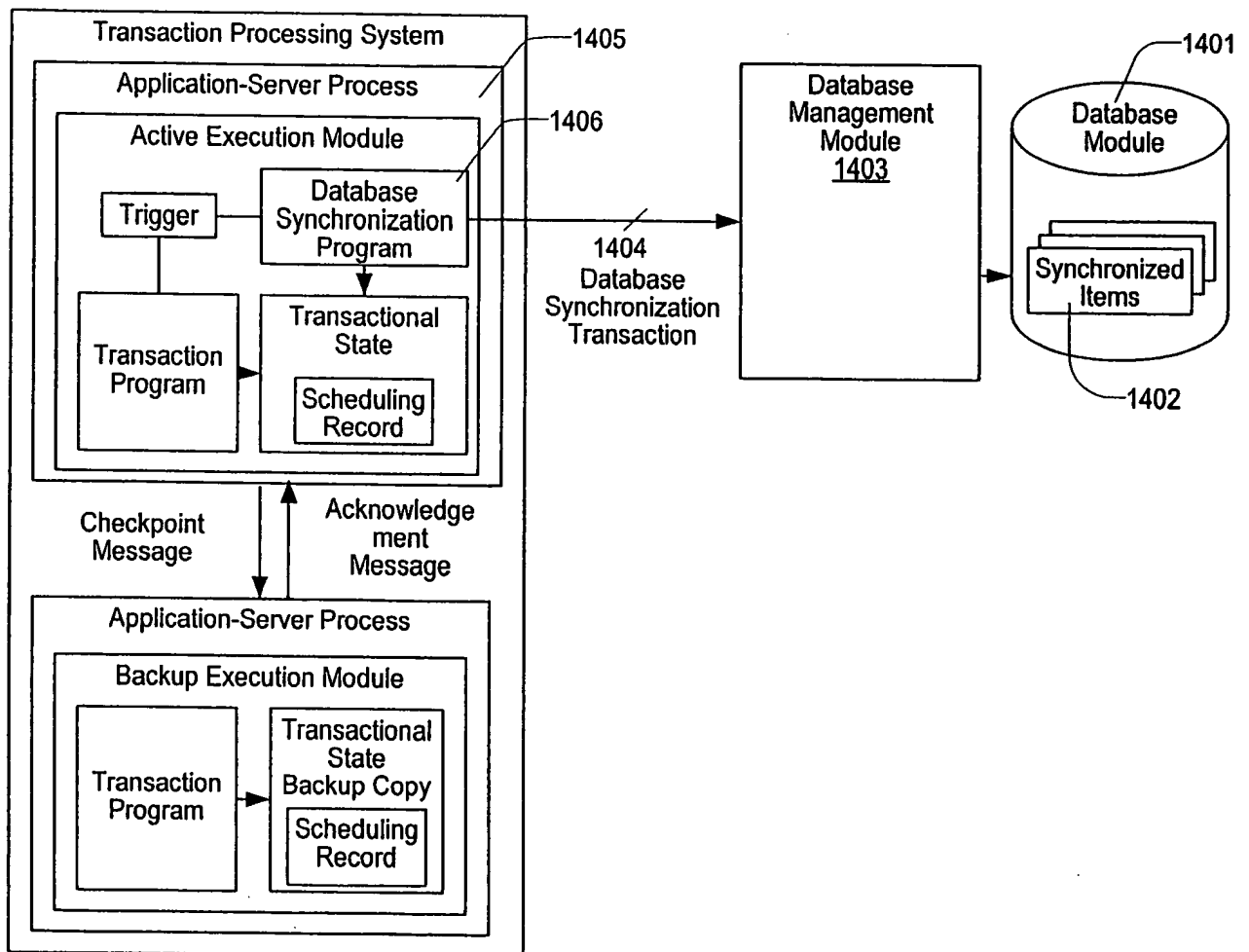


FIG. 14

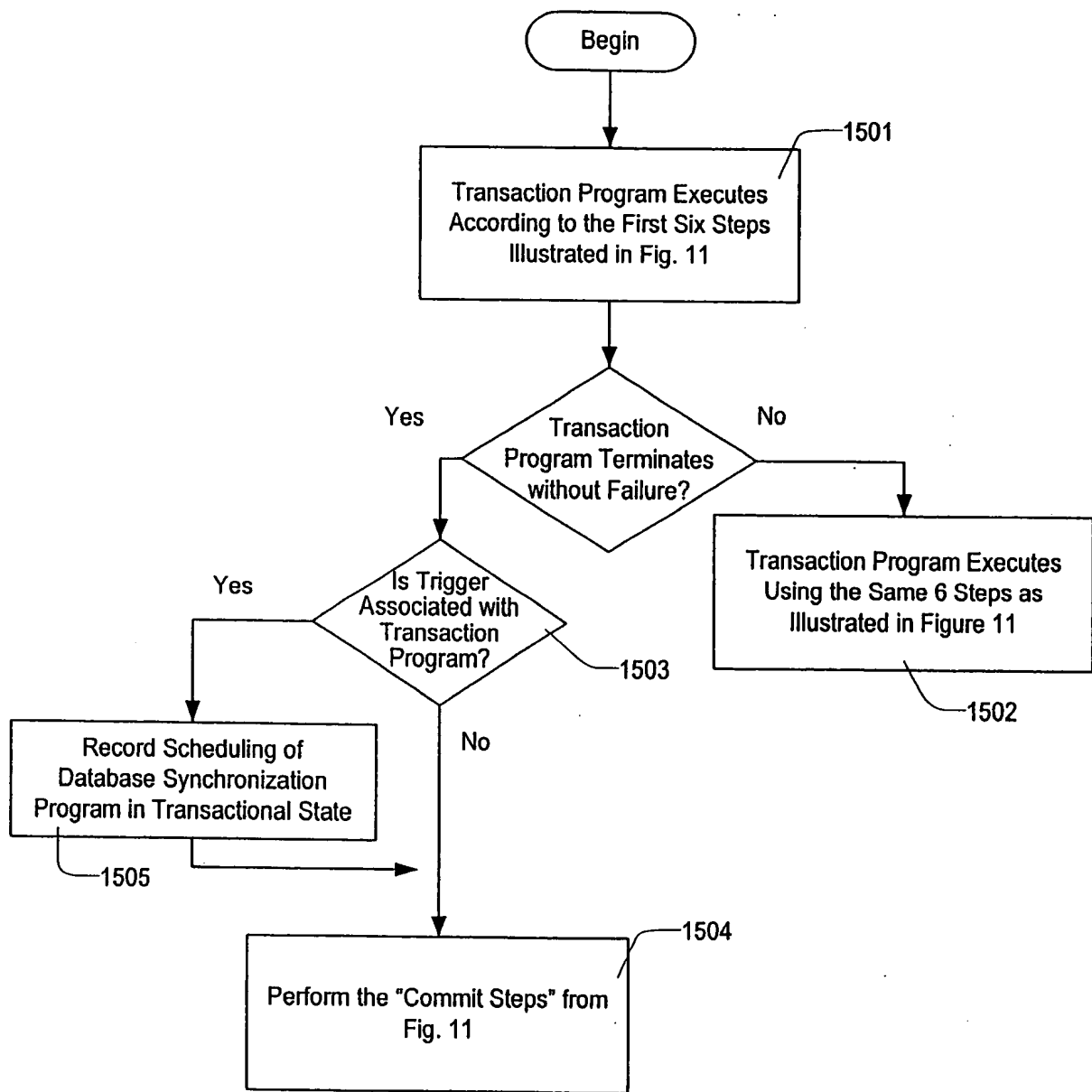


FIG. 15

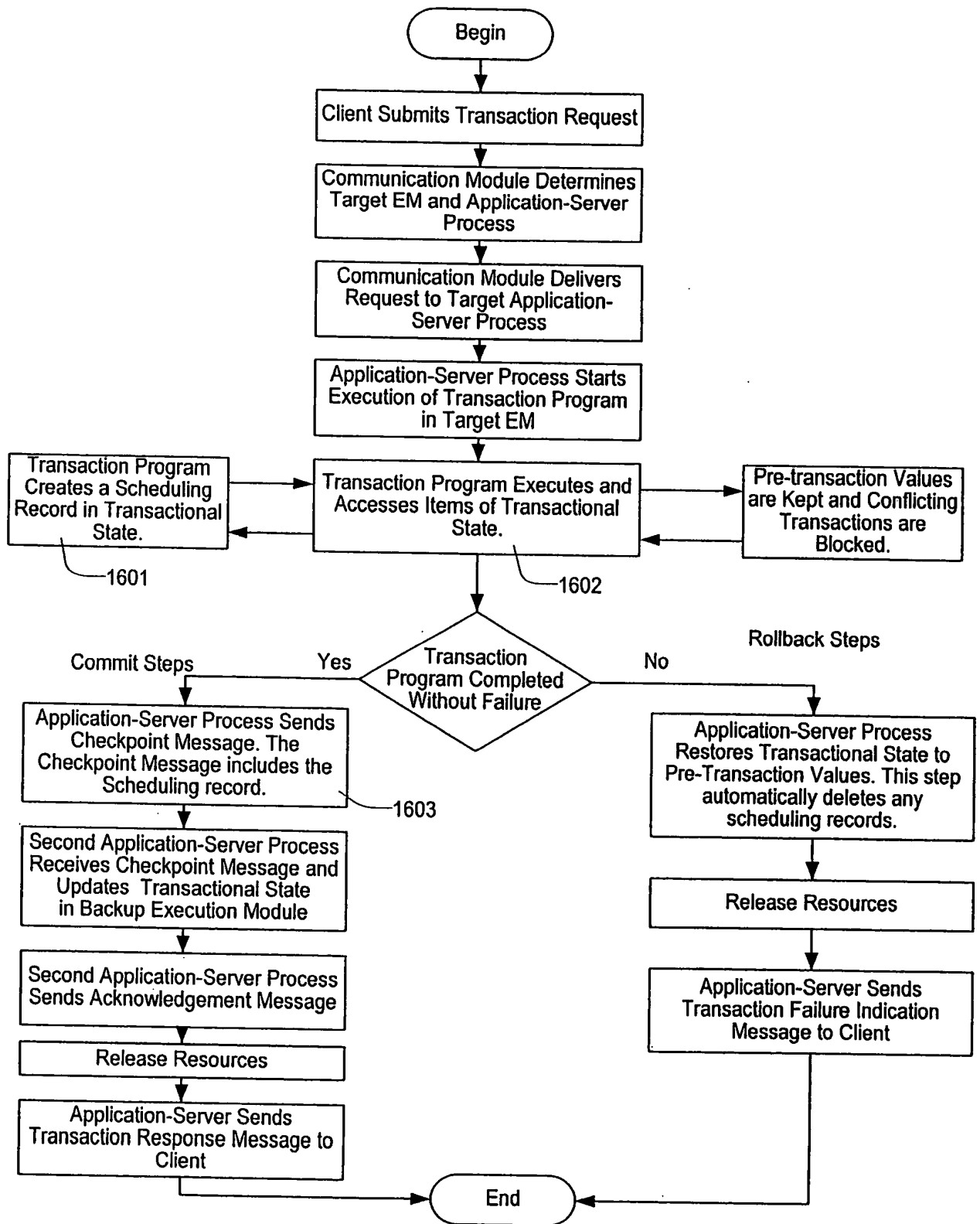


FIG. 16

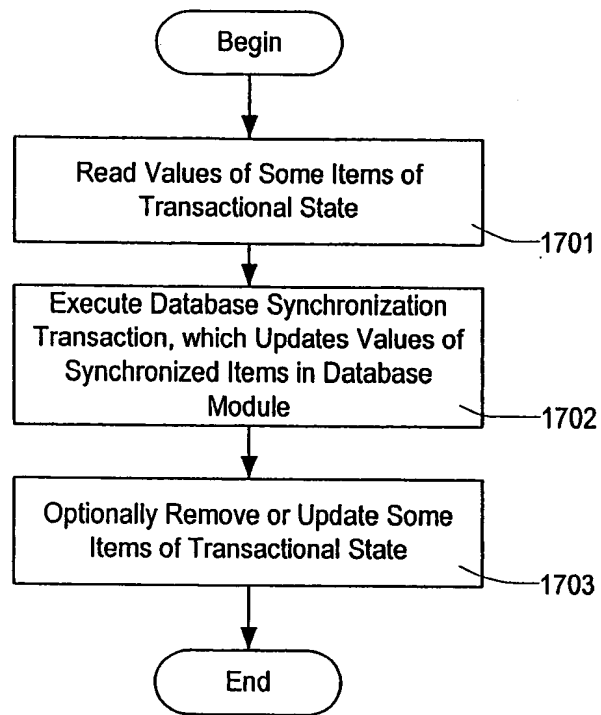


FIG. 17



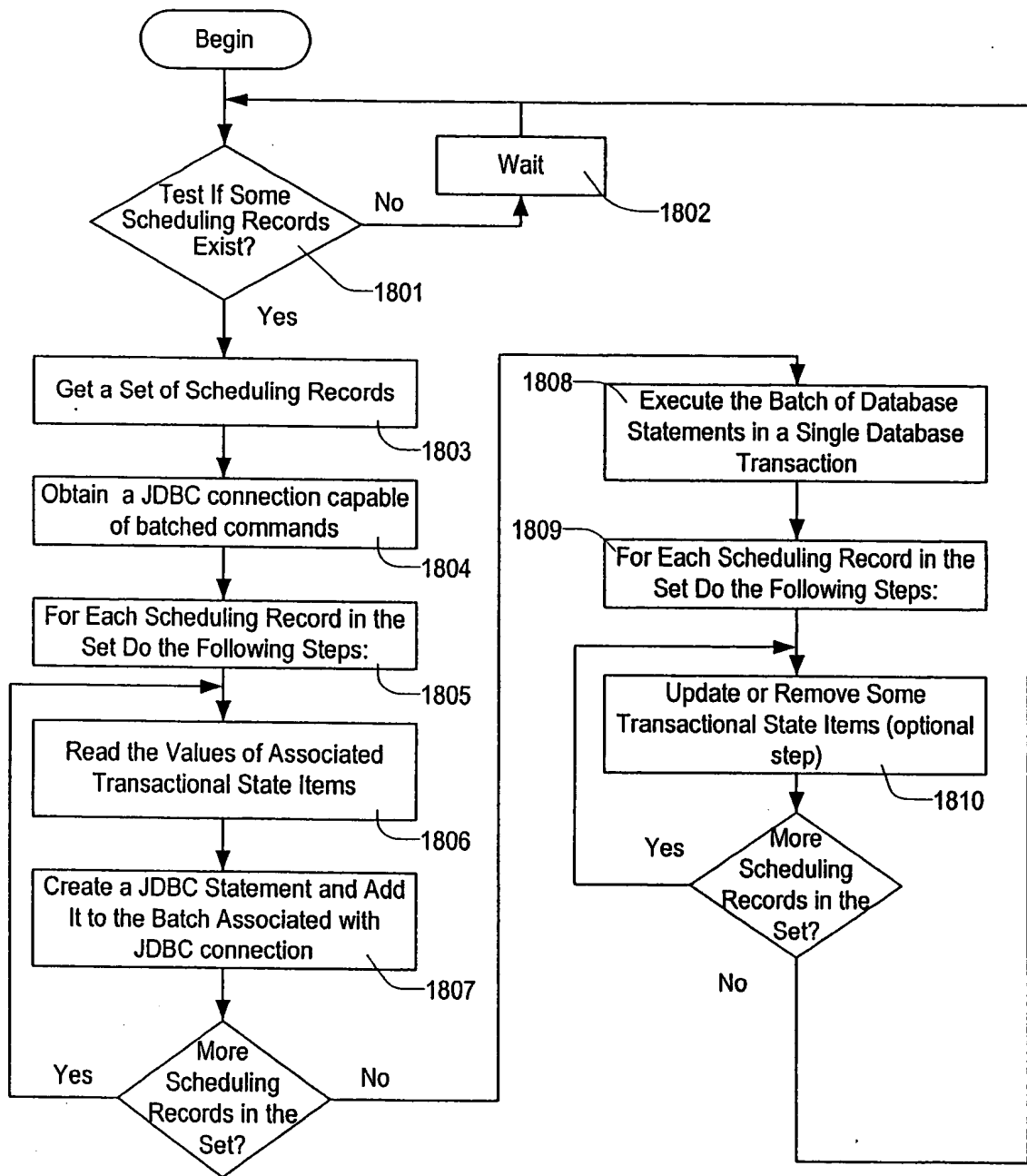
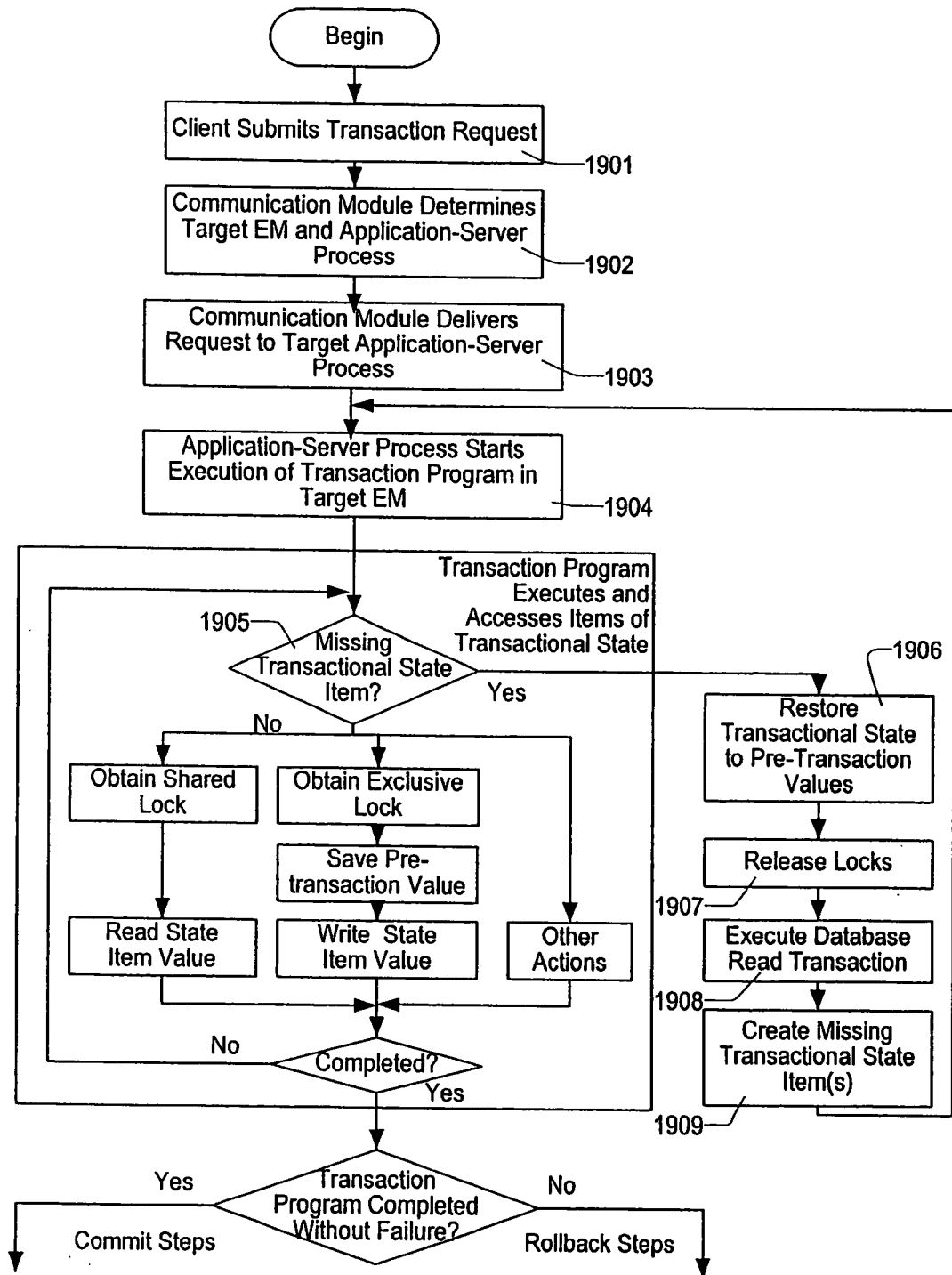


FIG. 18



Same Steps As in Fig. 12 or Fig. 13

FIG. 19

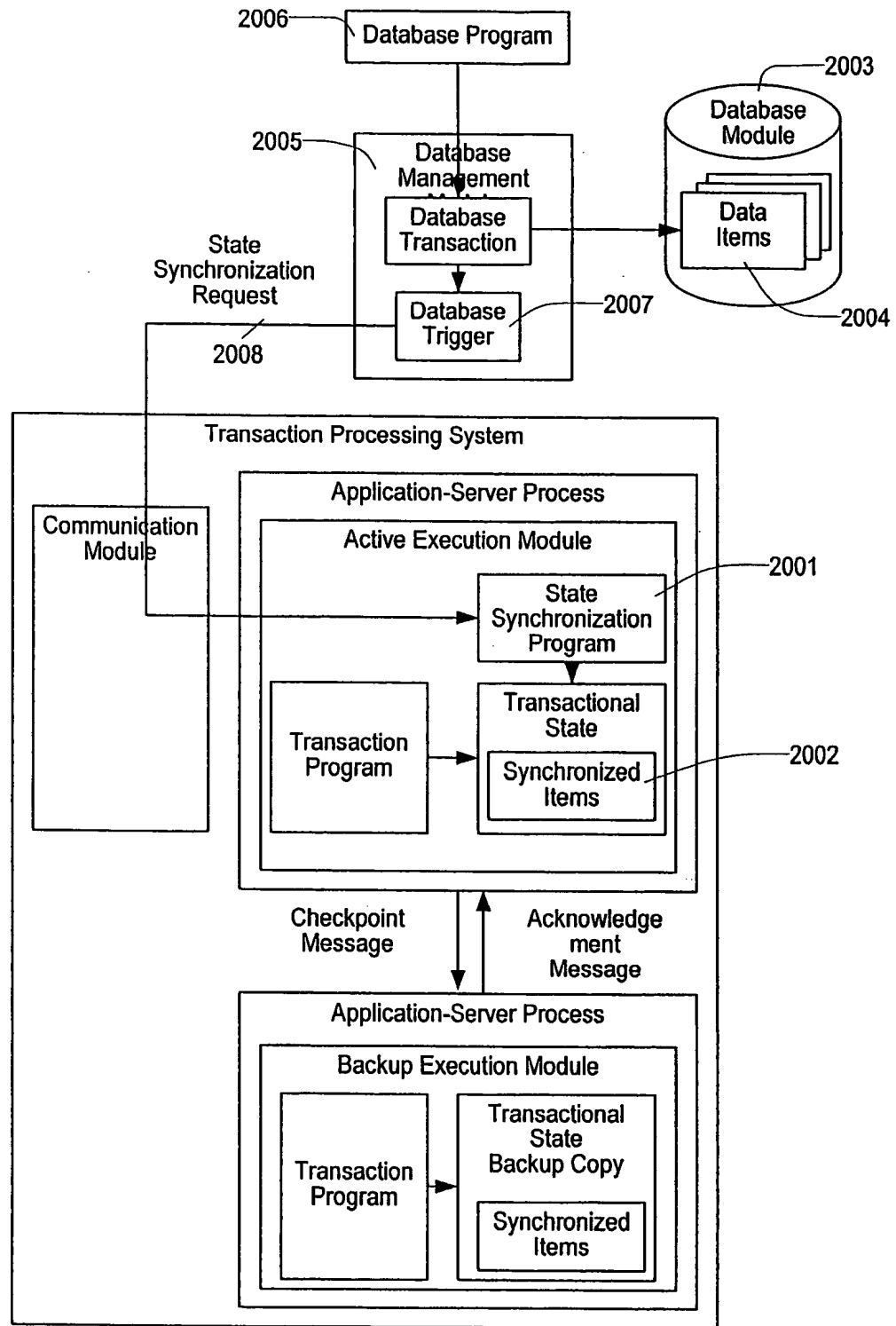


FIG. 20

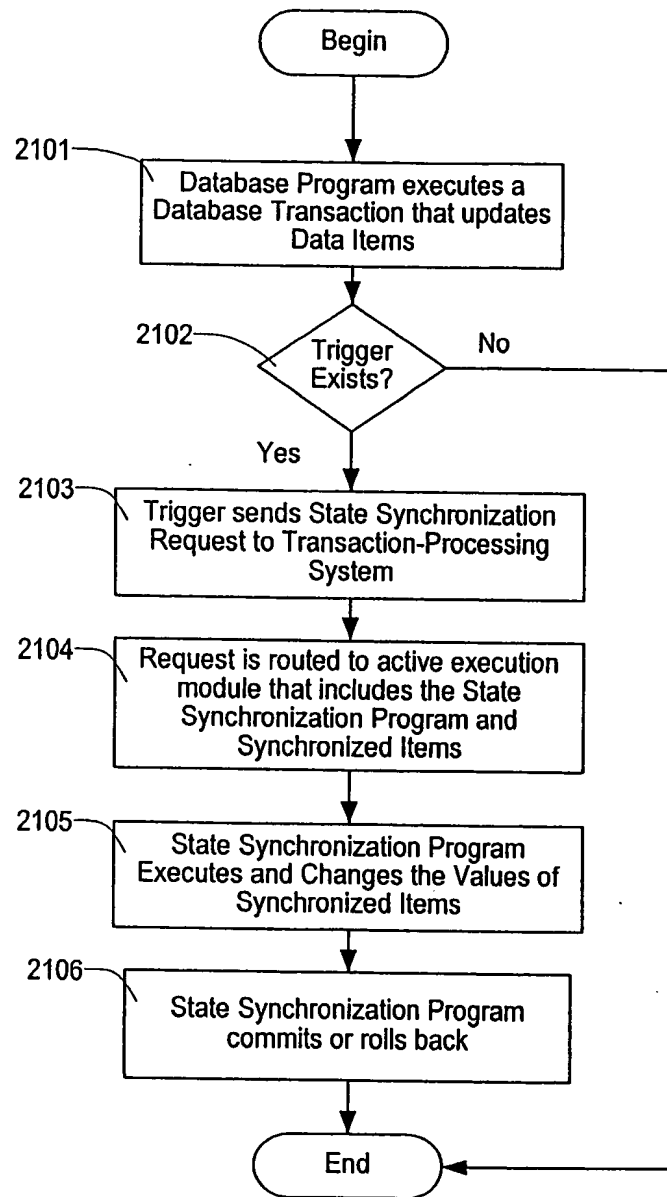


FIG. 21

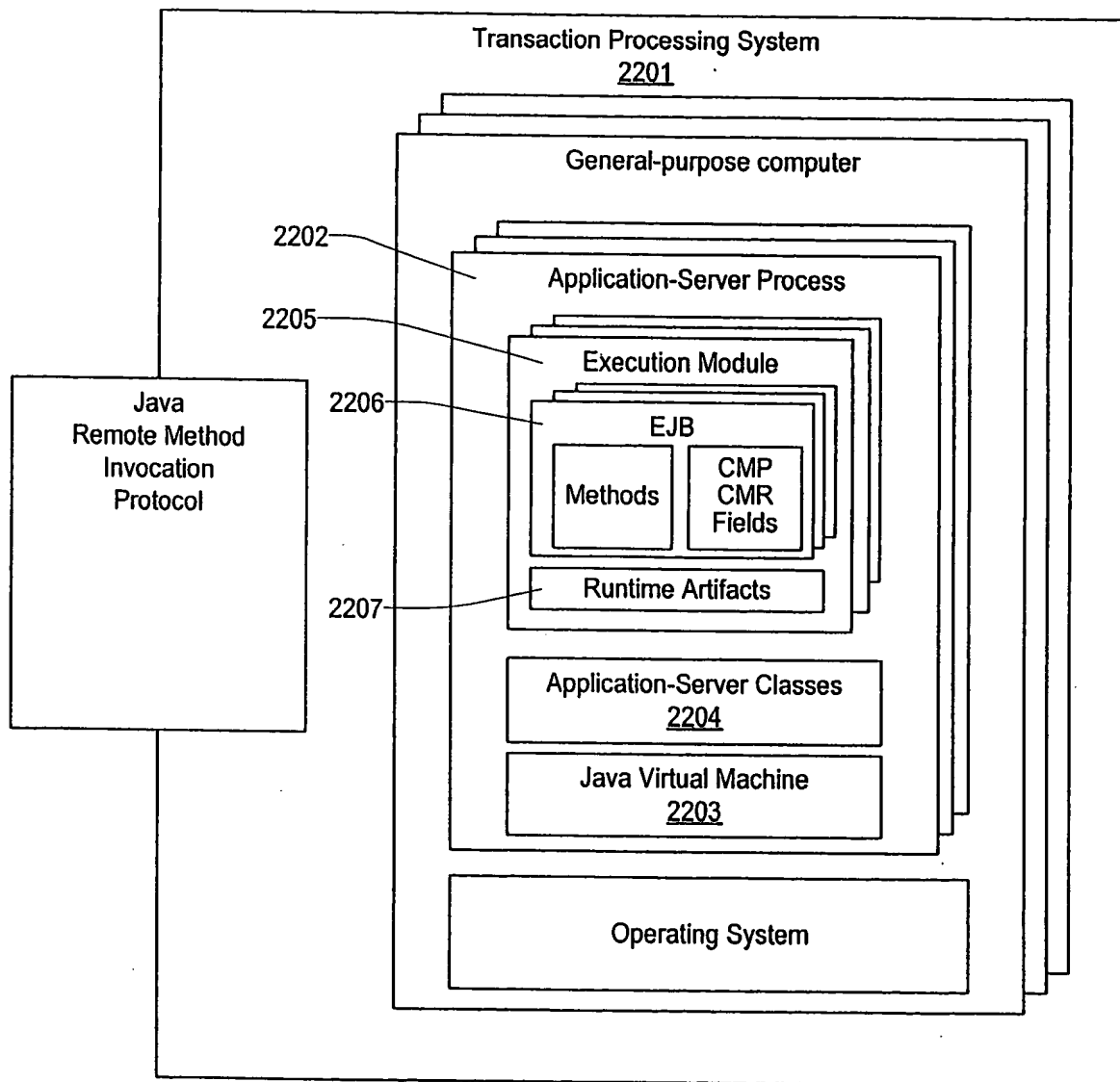


FIG. 22

```

public abstract class AccountBean implements EntityBean {
    public abstract String getAccountNumber();
    public abstract void setAccountNumber();

    public abstract double getBalance();
    public abstract void setBalance(double newBalance);

    public void debit(double amount) {
        setBalance(getBalance() - amount);
    }

    public void credit(double amount) {
        setBalance(getBalance() + amount);
    }

    // ...
}

```

FIG. 23

```

class AccountBeanImpl extends AccountBean {

    private double _balance;
    private double _balance_preTx;
    private boolean _balance_modified;

    public double getBalance() {
        return _balance;
    }

    public void setBalance(double newBalance) {
        if (!_balance_modified) {
            _balance_modified = true;
            _balance_preTx = _balance;
        }
        _balance = newBalance;
    }

    // ... similar code for accountNumber
}

```

FIG. 24

```

public class Account {
    private String accountNumber;
    private double balance;

    public void debit(double amount) {
        balance -= amount;
    }

    public void credit(double amount) {
        balance += amount;
    }
}

```

FIG. 25

```

public interface Account extends EJBLocalObject {

    // CMP field accountNumber
    public String getAccountNumber();

    // CMP field balance
    public double getBalance();
    public void setBalance(double newBalance);

    // EJB business methods.
    public void debit(double amount);
    public void credit(double amount);
}

```

FIG. 26

```

public abstract class AccountBeanTrigger extends AccountBean {
    AccountSynchronizationHome accountSynchronizationHome;

    public void debit(double amount) throws EJBException {
        super.debit(amount);
        try {
            accountSynchronizationHome.create(getAccountNumber());
        } catch (DuplicateKeyException e) {
            // This exception indicates that a database
            // synchronization program for this
            // account number is already scheduled.
        } catch (Exception e) {
            // Unexpected error.
            throw new EJBException(e);
        }
    }
}

```

FIG. 27

```

public class AccountBeanImpl implements AccountBeanTrigger {
    // same code as in Fig. 24
}

```

FIG. 28

```

public interface DatabaseSynchronization extends EJBLocalObject {
    void readValues() throws Exception;
    void updateDatabase() throws Exception;
    void removeItems() throws Exception;
}

```

FIG. 29

```

public interface AccountSynchronization extends DatabaseSynchronization {
}

```

FIG. 30

```

public interface AccountSynchronizationHome extends EJBLocalHome {
    AccountSynchronization create(String accountNumber)
        throws CreateException;
    AccountSynchronization findByPrimaryKey(String accountNumber)
        throws FinderException;
}

```

FIG. 31



```

public abstract class AccountSynchronizationBean implements EntityBean {

    EntityContext entityContext;
    AccountHome accountHome;

    public abstract String getAccountNumber();
    public abstract void setAccountNumber(String accountNumber);

    private Account account;

    // Values of read fields.
    private String accountNumber;
    private double latestBalance;

    public String ejbCreate(String accountNumber) {
        setAccountNumber(accountNumber);
        account = accountHome.findByPrimaryKey(accountNumber);
        return null;
    }

    public void readValues() throws Exception {
        accountNumber = getAccountNumber();
        latestBalance = account.getBalance();
    }

    public void updateDatabase() throws Exception {
        Connection conn = ...; // obtain a JDBC connection;
        PreparedStatement st = conn.prepareStatement(
            "UPDATE Account SET balance = ? WHERE accountNumber = ?");

        st.setDouble(1, latestBalance);
        st.setString(2, accountNumber);

        if (st.executeUpdate() < 1) {
            throw new Exception("Account " + accountNumber + " does not exist");
        }
    }

    public void removeItems() throws Exception {
        if (account.getBalance() == latestBalance) {
            entityContext.getEJBLocalObject().remove();
        } else {
            // Retain this EJB because another synchronization transaction
            // is necessary.
        }
    }

    // ...
}

```

FIG. 32

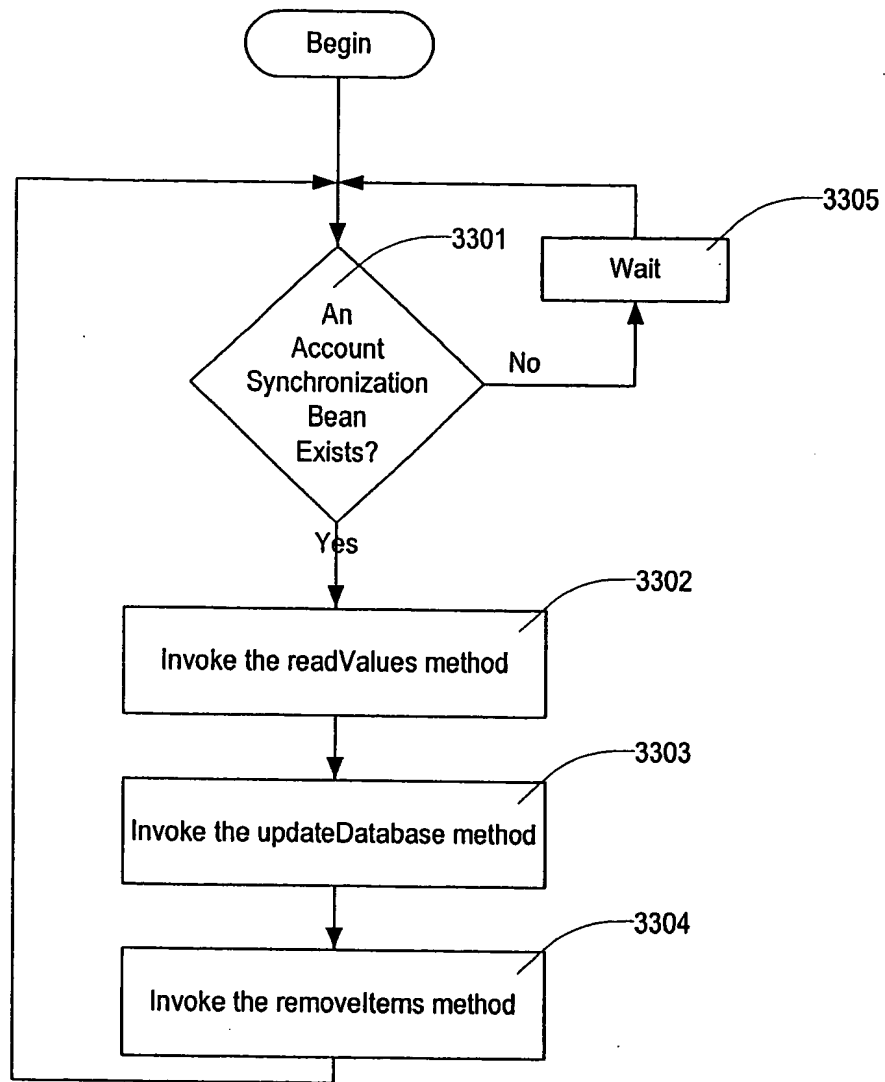


FIG. 33

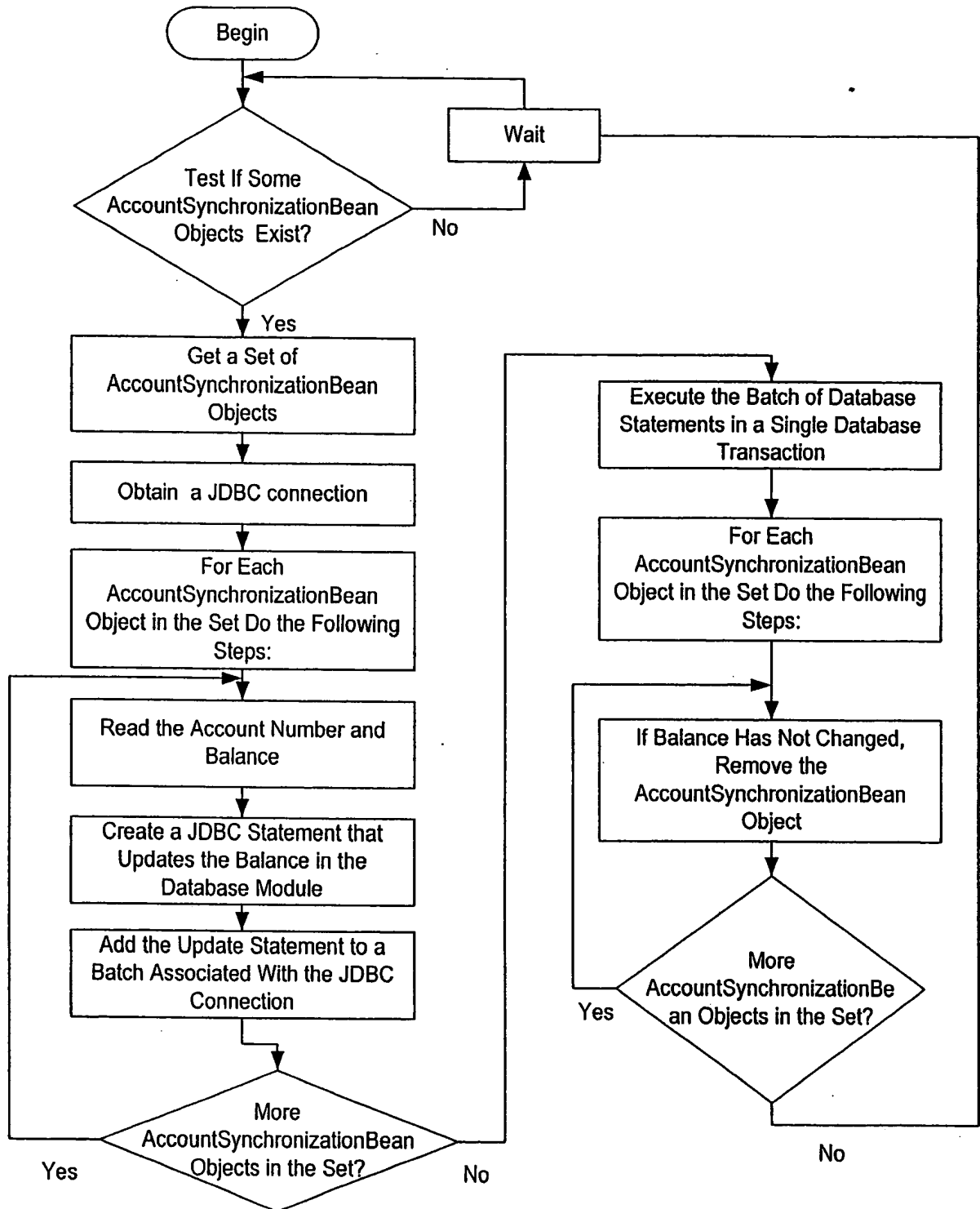


FIG. 34

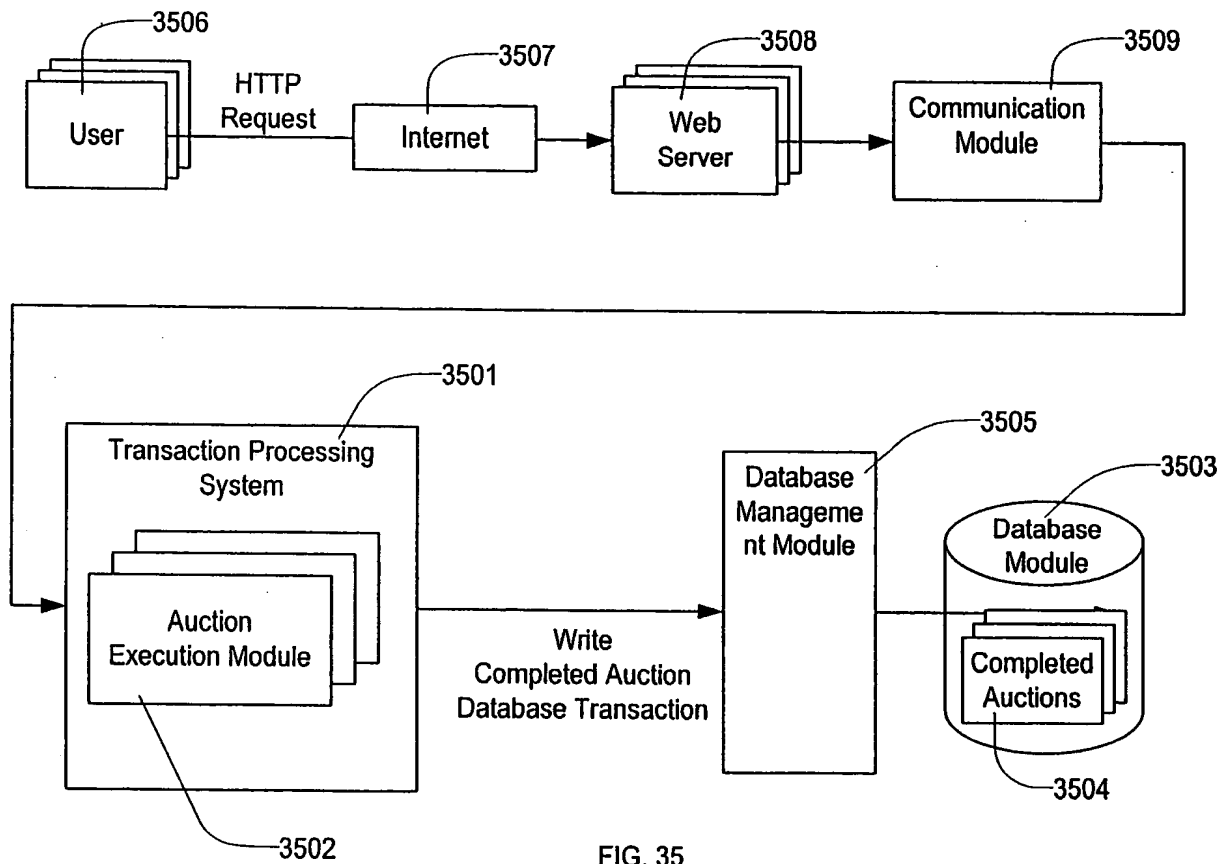


FIG. 35

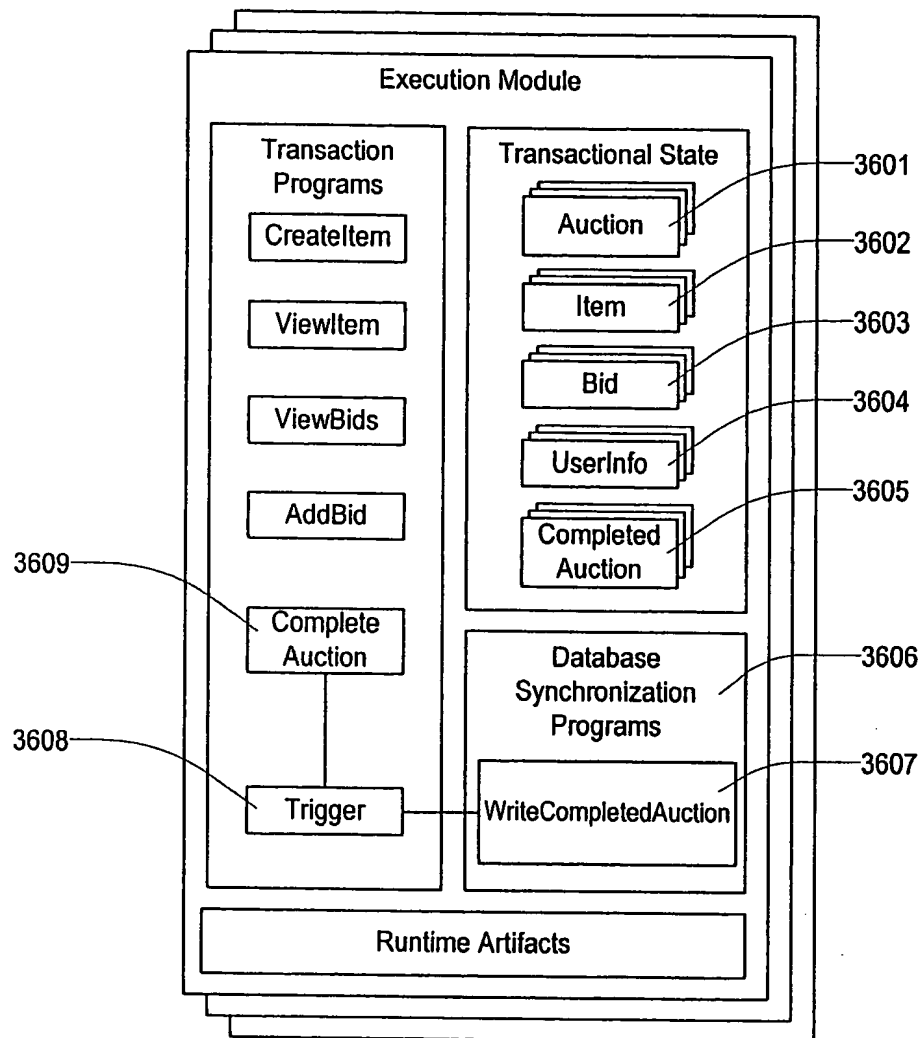


FIG. 36

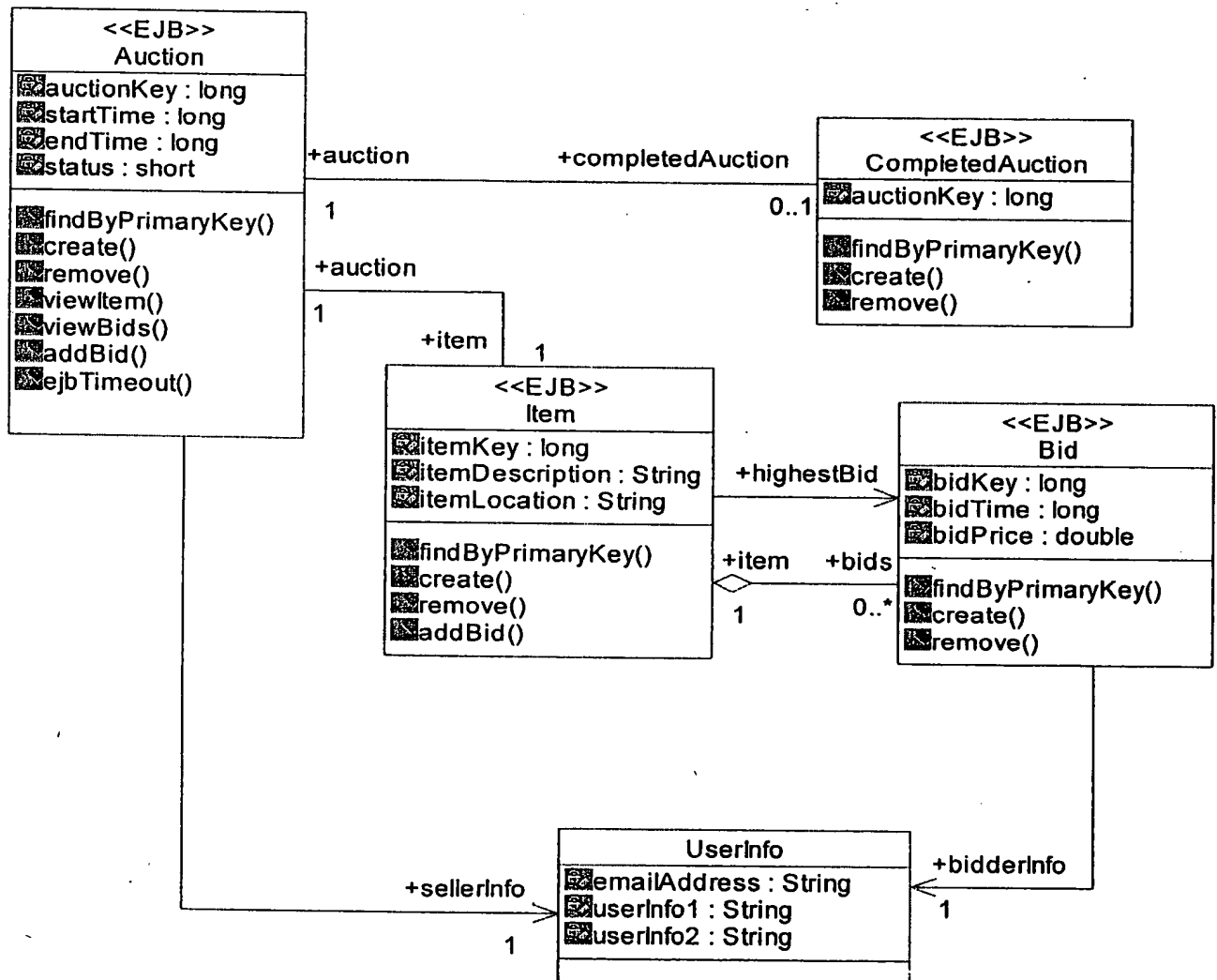


FIG. 37

```

/**
 * Enterprise JavaBean implementing an auction.
 */
abstract public class AuctionBean implements EntityBean, TimedObject {
    EntityContext entityContext;
    UserHome userHome;
    ItemHome itemHome;
    // EJB CMP and CMR fields accessor methods
    public abstract void setAuctionKey(Long auctionKey);
    public abstract Long getAuctionKey();
    public abstract void setItem(Item item);
    public abstract void setStartTime(long startTime);
    public abstract void setEndTime(long endTime);
    public abstract void setStatus(short status);
    public abstract void setCompletedAuction(CompletedAuction completedAuction);
    public abstract void setSellerInfo(UserInfo sellerInfo);
    public abstract short getStatus();
    public abstract CompletedAuction getCompletedAuction();
    public abstract UserInfo getSellerInfo();
    public abstract long getEndTime();
    public abstract long getStartTime();
    public abstract Item getItem();
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void unsetEntityContext() { entityContext = null;}
    public void setEntityContext(EntityContext entityContext) {
        this.entityContext = entityContext;
        userHome = ...;
        itemHome = ...;
    }

    /** Creates an auction for an item. */
    public Long ejbCreate(UserInfo sellerInfo, CreateItemInfo itemCopy, CategoryRemote category)
        throws CreateException, AuctionException {
        setAuctionKey(new Long(KeyGen.getNextKey()));
        setStartTime(itemCopy.getStartTime());
        setEndTime(itemCopy.getEndTime());
        setSellerInfo(sellerInfo);
        itemCopy.setCategory(category);
        return null;
    }

    public void ejbPostCreate(UserInfo sellerInfo, CreateItemInfo itemCopy, CategoryRemote category)
        throws CreateException, AuctionException {
        try {
            itemHome.create(new Long(KeyGen.getNextKey()),
                (Auction) entityContext.getEJBLocalObject(), itemCopy);
        } catch (CreateException ex) {
            entityContext.setRollbackOnly();
            throw ex;
        }
    }
}

```

FIG. 38

Code continues in Fig. 39

Continuation of code in Fig. 38

```
        TimerService timerService = entityContext.getTimerService();
        long endTime = itemCopy.getEndTime();
        long currentTime = System.currentTimeMillis();
        long duration = endTime - currentTime;
        timerService.createTimer(duration, null);
    }

    /** Removes this auctioned item. */
    public void ejbRemove() throws RemoveException {
        getItem().remove();
    }

    public double getHighestBid() throws AuctionException {
        return getItem().getHighestBid();
    }

    /** Obtains the current information about this auctioned item. */
    public ViewItemResult viewItem() throws AuctionException {
        Item item = getItem();
        ViewItemResult copy = new ViewItemResult();
        copy.setDescription(item.getItemDescription());
        copy.setStartTime(getStartTime());
        copy.setEndTime(getEndTime());
        copy.setLocation(item.getItemLocation());
        copy.setItemKey(getAuctionKey().longValue());
        copy.setHighestBid(item.getHighestBid());
        copy.setNumberOfBids(item.getBids().size());
        return copy;
    }

    /** Adds a bid on this auctioned item. */
    public void addBid(UserInfo bidderInfo, double bid) throws AuctionException {
        getItem().addBid(bidderInfo, bid);
    }

    /** Obtains all bids on this auctioned item. */
    public ViewBidsResult viewBids() throws AuctionException {
        return new ViewBidsResult(getItem().getBids());
    }

    /** Handles the end of auction timer invocation. */
    public void ejbTimeout(javax.ejb.Timer timer) {
        try {
            LocalHomes.getCompletedAuctionHome().create((Auction) entityContext.getEJBLocalObject());
        } catch (CreateException e) {
            logger.log(Level.WARNING, "unexpected CreateException", e);
        }
    }
}
```

FIG. 39



```

/**
 * Background thread that periodically writes completed auctions to a database.
 */
public class DatabaseUpdateThread extends Thread {
    static final int maxRowsPerInsert = 64; // Maximum batch size
    UserTransaction ut = ...; // Obtain the UserTransaction interface
    CompletedAuctionHome h = ...; // Obtain EJB local home interface

    public void run() {
        while (true) {
            try {
                Thread.sleep(1000); // Sleep 1 second
                writeCompletedAuctions();
            } catch (InterruptedException e) {}
        }
    }

    private void writeCompletedAuctions() {
        CompletedAuctionHome h = LocalHomes.getCompletedAuctionHome();
        ArrayList keyArrayList;
        // Obtain a ArrayList with a copy of all completed auction keys.
        try {
            ut.begin(); // This an EJB, not JDBC, transaction
            keyArrayList = new ArrayList(h.obtainAllKeys());
            ut.commit();
            txInProgress = false;
        } catch (Exception e) {
            // Handle exceptions
        }

        // If keyArrayList is too big, break the database updates into multiple smaller batches.
        int totalCount = keyArrayList.size();
        int processedCount = 0;
        while (processedCount < totalCount) {
            int thisBatchCount = Math.min(maxRowsPerInsert, totalCount - processedCount);
            processBatch(keyArrayList.subList(processedCount, processedCount + thisBatchCount));
            processedCount += thisBatchCount;
        }
    }

    /** Executes a batch using a single database transaction. */
    private void processBatch(List keyArrayList) {
        Connection con = null; // Code continues in Fig. 41
        PreparedStatement stmt = null;
        SQLException sqlException = null;
        try {
            con = ...; // Obtain a JDBC connection

```

FIG. 40

Code continues in Fig. 41

Continuation of code in Fig. 40

```

// Read all completed transactions in this Auction EM in one read-only EJB transaction.
stmt = con.prepareStatement("INSERT INTO COMPLETED_AUCTIONS VALUES(?,?,?,?,?,?,?,?)");
try {
    ut.begin();
    // This is an EJB, not JDBC, transaction
    for (Iterator it = keyArrayList.iterator(); it.hasNext(); ) {
        Long key = (Long) it.next();
        CompletedAuction completedAuction = h.findByPrimaryKey(key);
        Auction auction = completedAuction.getAuction();
        Item item = auction.getItem();
        UserInfo sellerInfo = auction.getSellerInfo();
        UserInfo buyerInfo = item.getHighestBid().getBidderInfo();

        stmt.setLong(1, item.getItemKey());           // Item key
        stmt.setString(2, item.getItemDescription()); // Item description
        stmt.setString(3, item.getItemLocation());    // Item location
        stmt.setTimestamp(4, new Timestamp(auction.getStartTime())); // Auction start time
        stmt.setTimestamp(5, new Timestamp(auction.getEndTime())); // Auction end time
        stmt.setString(6, sellerInfo.getEmailAddress()); // Seller's key
        stmt.setString(7, buyerInfo.getEmailAddress()); // Buyer's key
        stmt.setDouble(8, item.getHighestBid());      // Final price

        stmt.addBatch();
    }
    ut.commit();
} catch (Exception e) {
    // Handle exception
}

// Execute the database insert batch outside of any EJB transaction to
// to prevent holding EJB locks across database access.
stmt.executeBatch();
con.commit();

// Release database connection here
} catch (SQLException e) {
    // Handle exception
}

/* Remove all CompletedAuctionBean objects that have been written to the database. */
try {
    ut.begin();
    for (Iterator it = keyArrayList.iterator(); it.hasNext(); ) {
        Long key = (Long) it.next();
        CompletedAuction completedAuction = h.findByPrimaryKey(key);
        completedAuction.remove();
    }
    ut.commit();
} catch (Exception e) {
    // Handle exception
}
}
}

```

FIG. 41

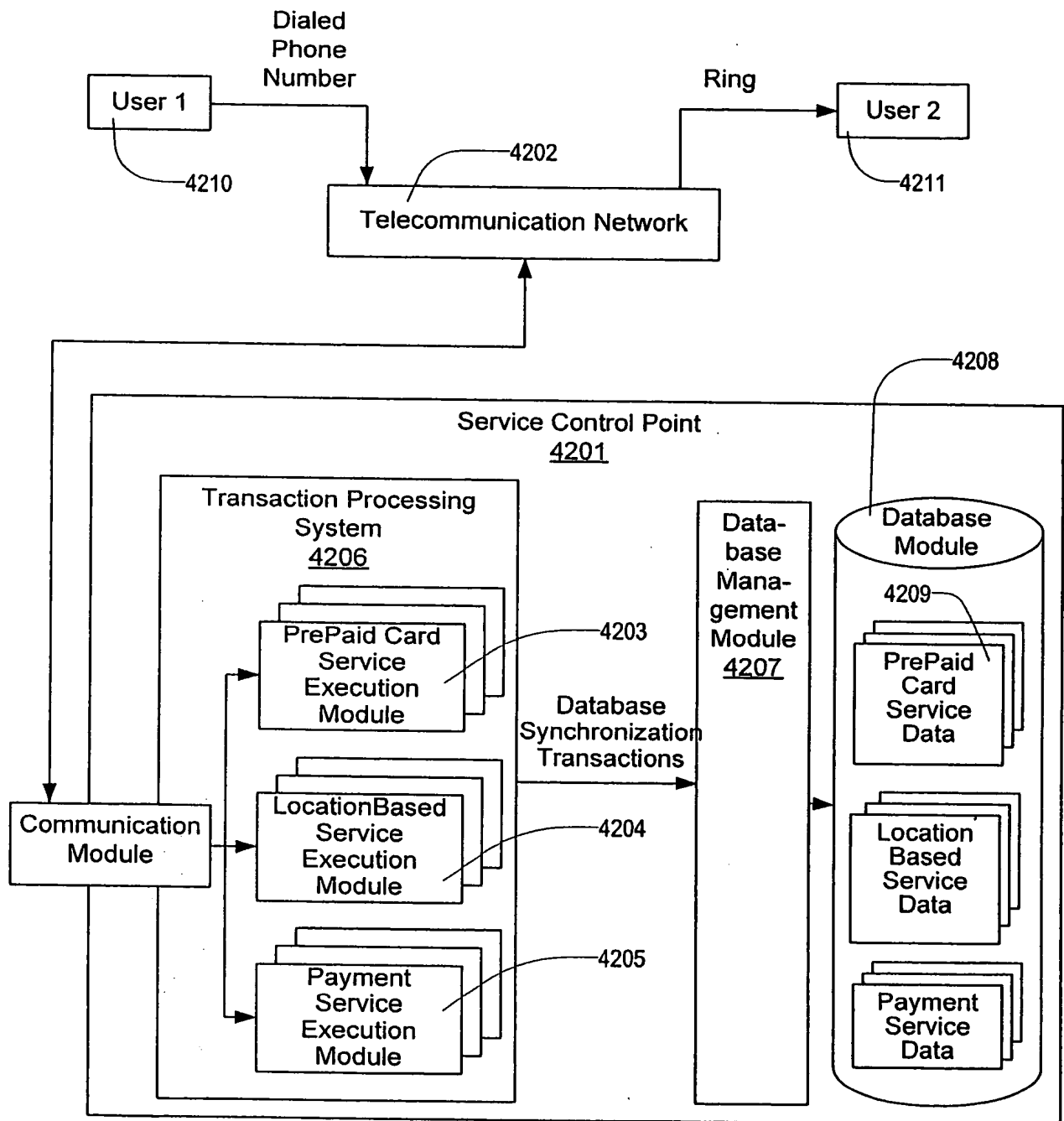


FIG. 42

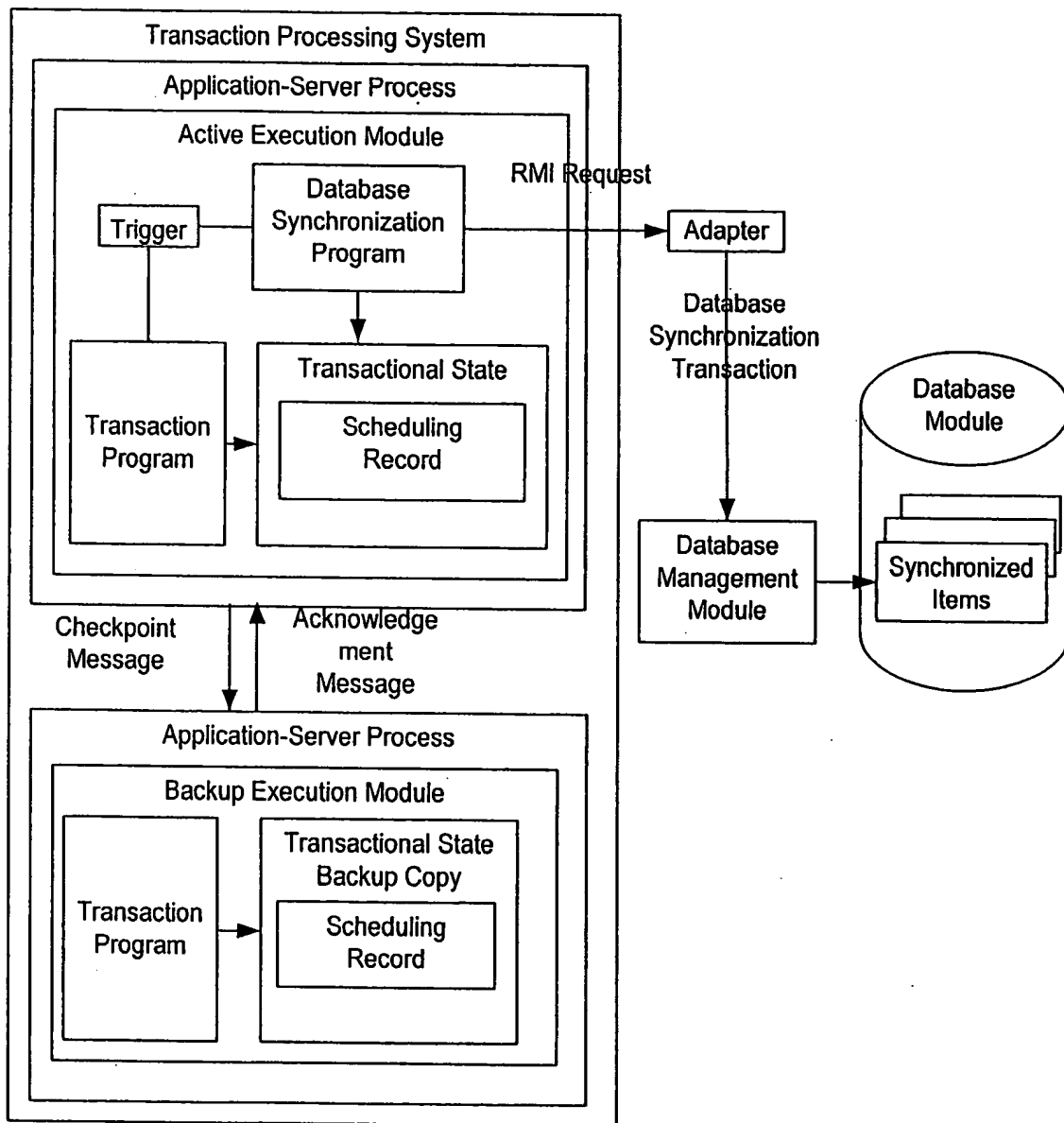


FIG. 43

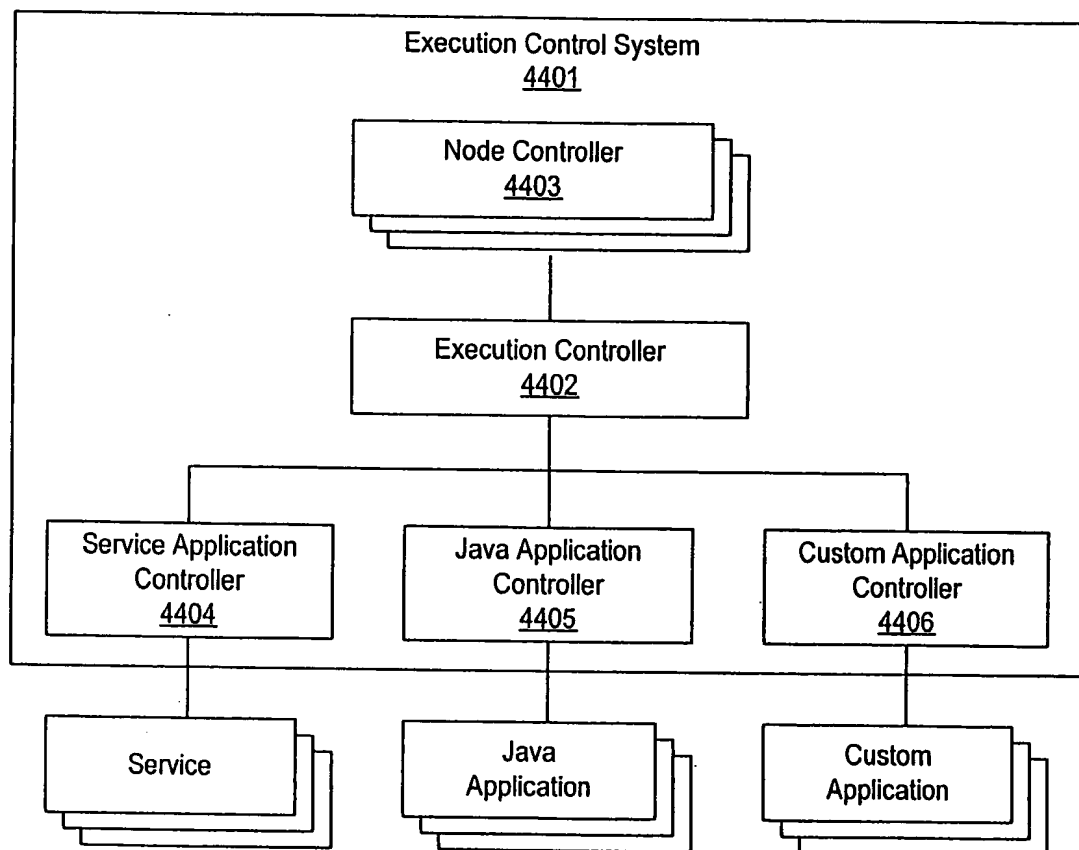


Fig 44

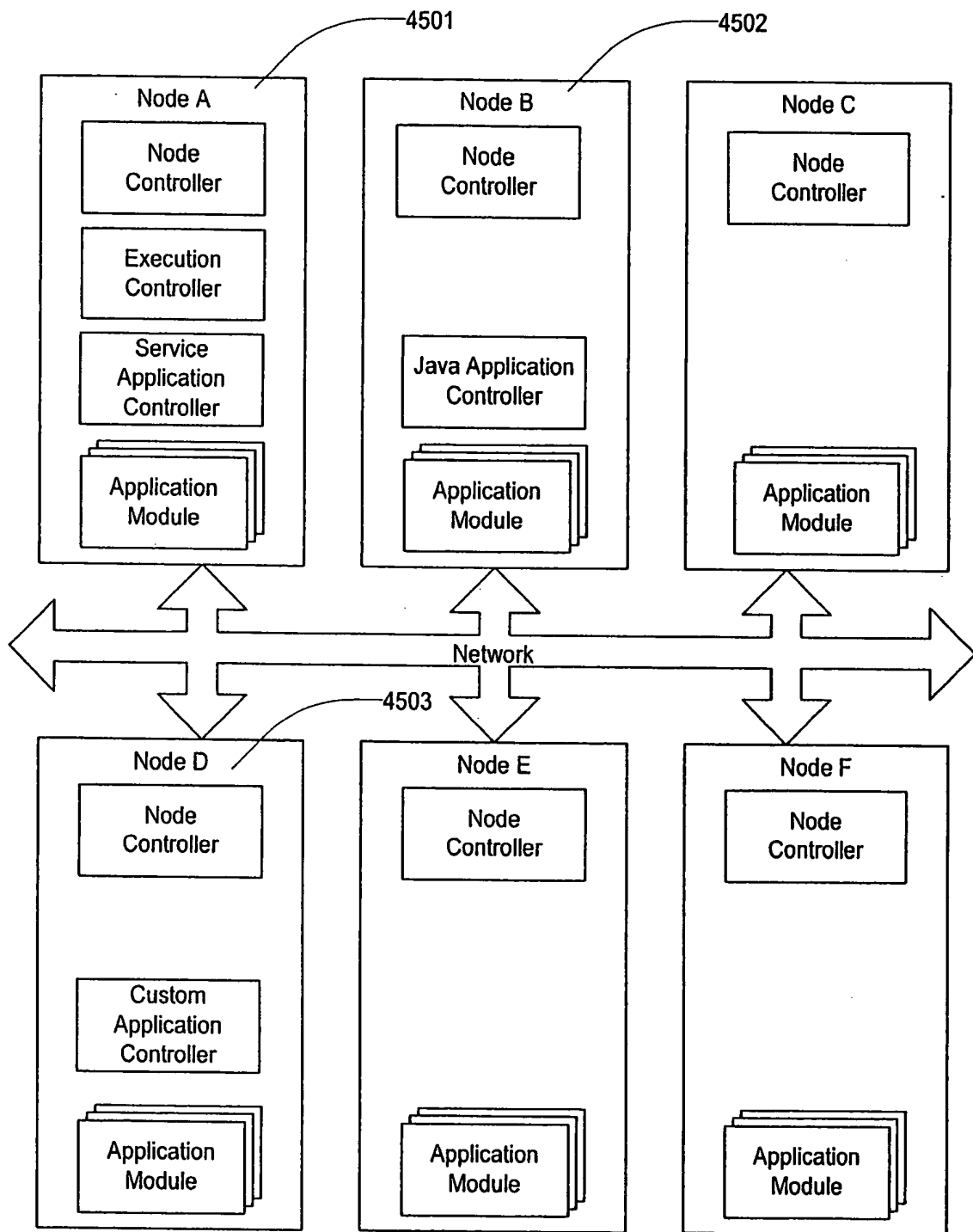


Fig 45

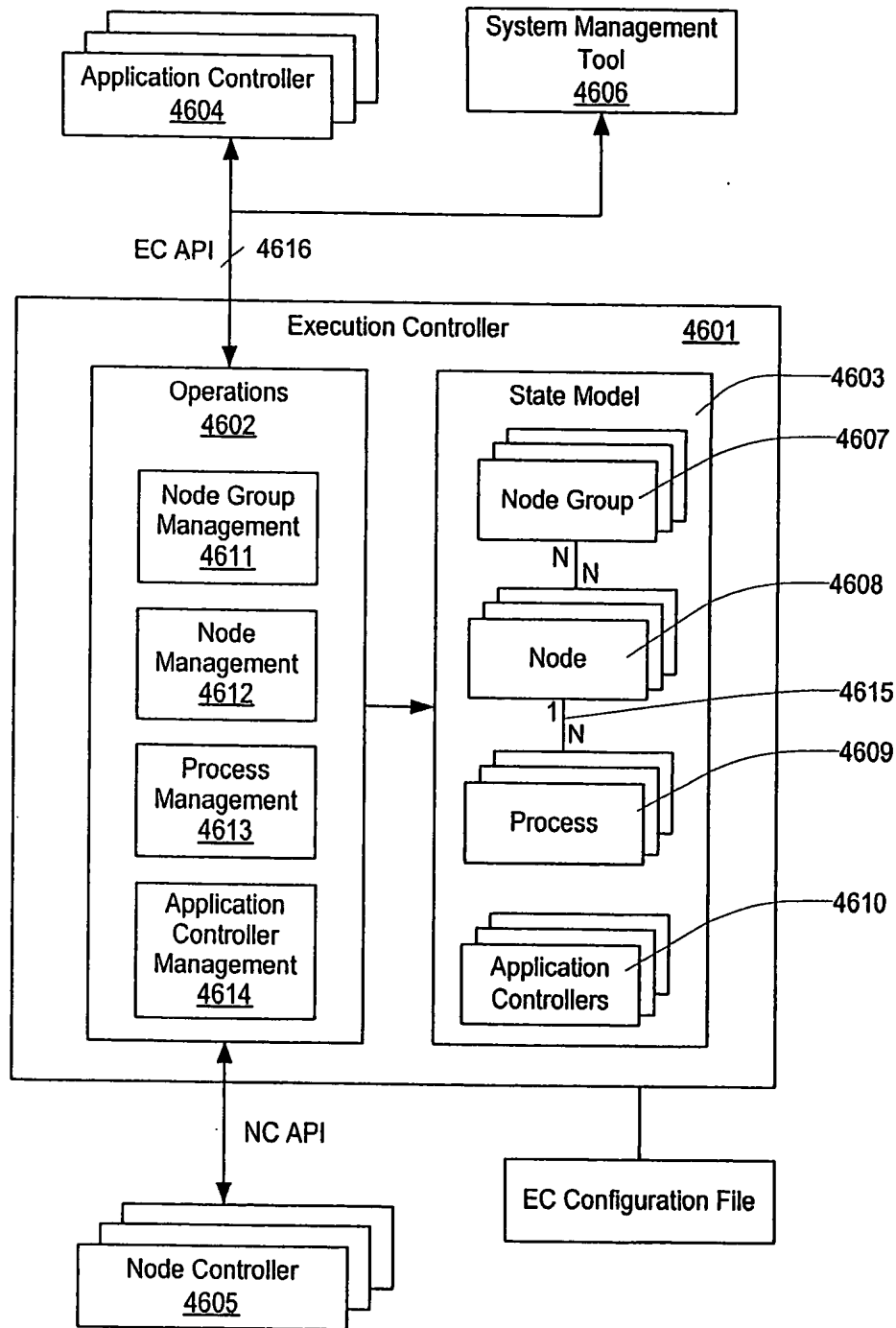


Fig 46

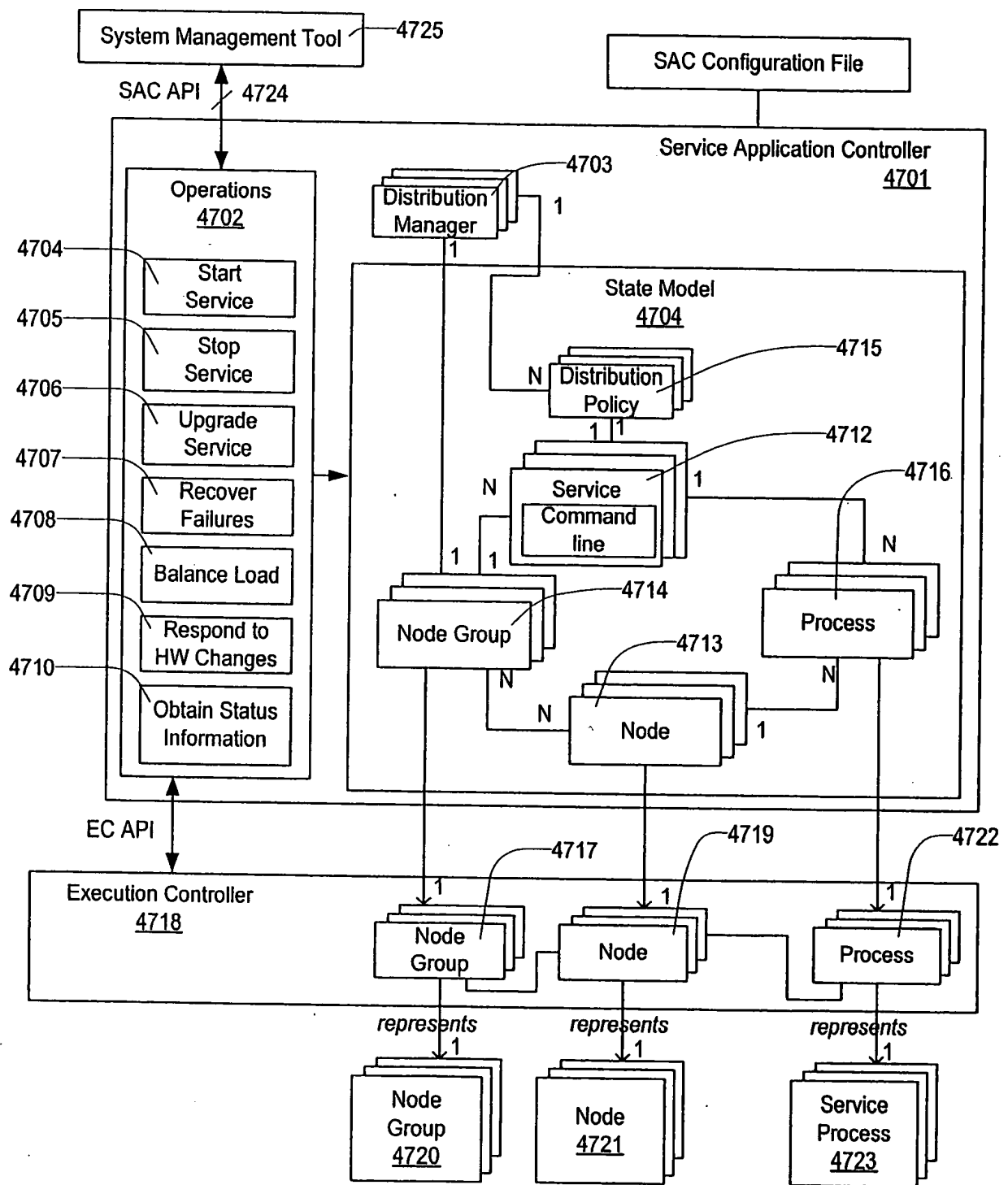


Fig 47



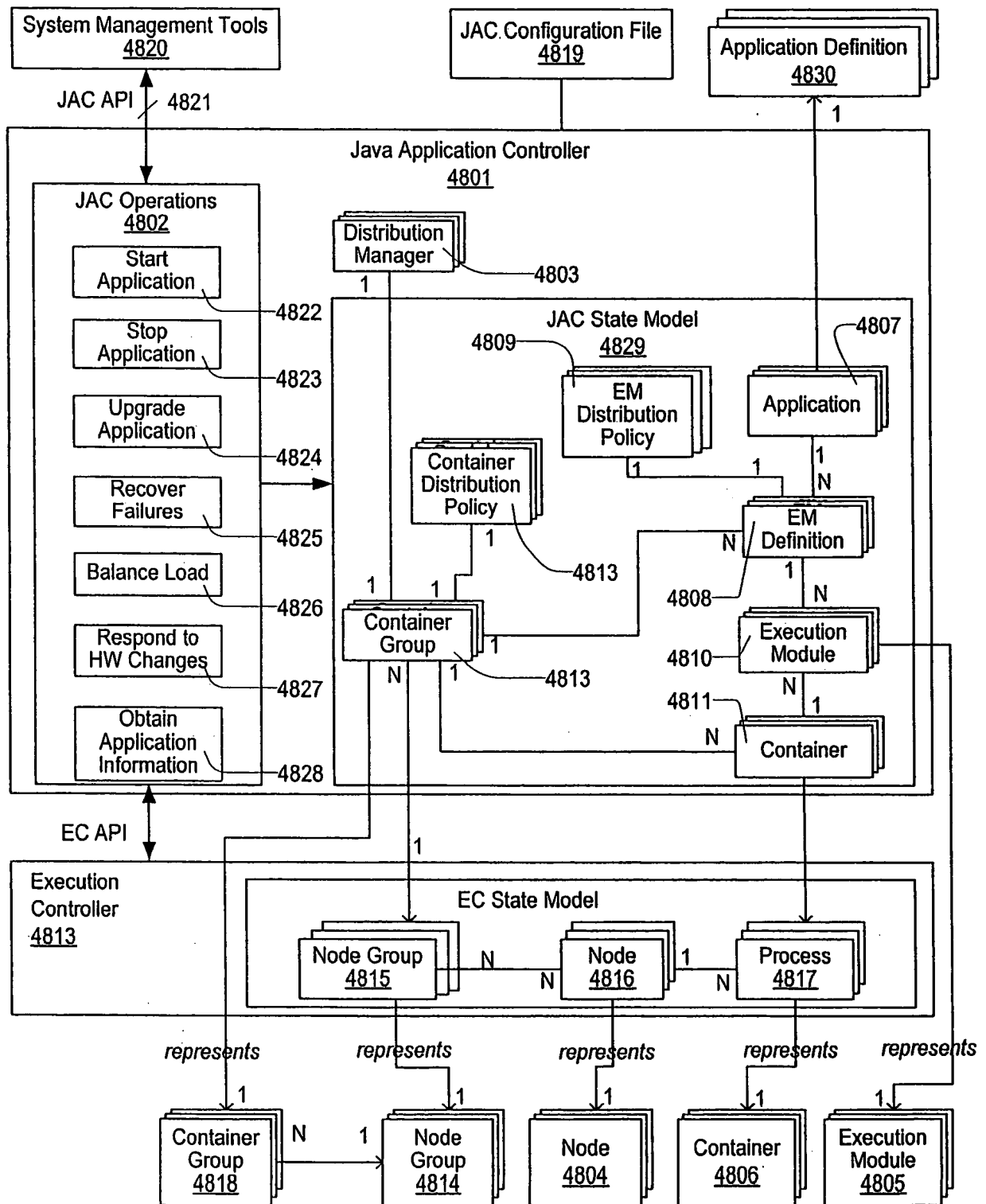


Fig 48